NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PORTING HIGH QUALITY GRAPHICS SIMULATIONS TO A LOW-COST COMPUTER ARCHITECTURE

by

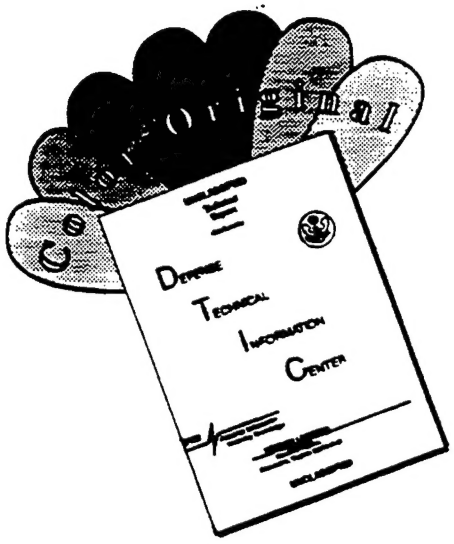
Jaime Borrego
and
Frank Free

September 1996

Thesis Advisors: David R. Pratt
John S. Falby

Approved for public release; distribution is unlimited.

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1996	Master's Thesis

4. TITLE AND SUBTITLE
PORTING HIGH QUALITY GRAPHICS SIMULATIONS TO A LOW-COST COMPUTER ARCHITECTURE

5. FUNDING NUMBERS

6. AUTHOR(S)
Borrego, Jaime
Free, Frank

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Two disadvantages of using Silicon Graphics®, Inc. (SGI) computers and SGI's IRIS® Performer™ application programming interface (API) in NPSNET are the current inability to run the graphic simulations on more popular environments, such as personal computer (PC) operating systems (OSs), and the increased expense associated with the alternative of choosing graphics specific hardware over lower cost PCs. Work detailed in this thesis addresses these problems by porting the graphics code from NPSNET to relatively inexpensive PC hardware running the Microsoft® Windows NT™ OS.

Two independent approaches were taken. The first created a library of graphics calls which simulate the syntax and functionality of Performer calls, but which have been redefined in terms of the Gemini Technology Corporation's OpenGVS™ API, which is capable of running on the NT platform. The second proposed and implemented a prototype graphics display manager coded using only OpenGVS, rather than Performer, for a proposed platform-independent redesign of NPSNET.

As a result of this effort, the goal of porting IRIS Performer graphics simulations to the PC has been accomplished, and a new architecture for NPSNET display managers has been validated.

14. SUBJECT TERMS	15. NUMBER OF PAGES
computer graphics, personal computer	273
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

PORTING HIGH QUALITY GRAPHICS SIMULATIONS TO A LOW-COST COMPUTER ARCHITECTURE

Frank Free
Major, United States Marine Corps
B.S., United States Naval Academy, 1983

Jaime Borrego
Lieutenant, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1996

Authors:

Frank Free

Jaime Borrego

Approved by:

David R. Pratt, Thesis Advisor

John S. Falby, Thesis Advisor

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Two disadvantages of using Silicon Graphics®, Inc. (SGI) computers and SGI's IRIS® Performer™ application programming interface (API) in NPSNET are the current inability to run the graphic simulations on more popular environments, such as personal computer (PC) operating systems (OSs), and the increased expense associated with the alternative of choosing graphics specific hardware over lower cost PCs. Work detailed in this thesis addresses these problems by porting graphics code from NPSNET to relatively inexpensive PC hardware running the Microsoft® Windows NT™ OS.

Two independent approaches were taken. The first created a library of graphics calls which simulate the syntax and functionality of Performer calls, but which have been redefined in terms of the Gemini Technology Corporation's OpenGVS™ API, which is capable of running on the NT platform. The second proposed and implemented a prototype graphics display manager coded using only OpenGVS, rather than Performer, for a proposed platform-independent redesign of NPSNET.

As a result of this effort, the goal of porting IRIS Performer graphics simulations to the PC has been accomplished, and a new architecture for NPSNET display managers has been validated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. BACKROUND

The NPSNET Research Group at the Computer Science Department of the Naval Postgraduate School in Monterey, California, is a group of faculty, staff and students working in various areas of networked virtual environments. The group's main software system is the Naval Postgraduate School Networked Vehicle Simulator (NPSNET), which integrates the research components into a networked computer simulation of up to 300 players using currently available off-the-shelf workstations and networking technology. [NPSN96]

NPSNET's networked virtual environment of interacting humans and ground, sea, and air vehicles has drawn interest from outside the research community as well. The funding made available by many US Government agencies, including the Defense Advanced Research Projects Agency (DARPA) and the US Army Research Laboratories (ARL), reflects the potential of virtual reality research to produce a safe and cost saving training alternative to placing the user in an actual physical training environment. Computer simulations may be more cost effective to run than fuel-consuming, maintenance-intensive ships, aircraft and tanks. Further, simulations may be capable of providing a safe environment for military operations rehearsal, complete with the added realism of a simulated enemy force, in an environment which, in reality, would be impossible to otherwise access due to prevailing political or strategic climates. [NPSN96]

NPSNET runs on any graphics workstation made by Silicon Graphics®, Inc. (SGI). The software was programmed using SGI's IRIS® Performer™ as the graphics application

programmer interface (API), a package of functions which takes advantage of both the special graphics hardware in SGI machines, and the graphics capabilities of SGI's IRIS GL™ API, on which it was based [NPSN96]. Although SGI computers and IRIS Performer have become a de facto standard in the production of computer graphics simulations and virtual reality (VR) programming due to their speed, ease of use, and technical capabilities, their proprietary nature renders software written for them unportable to other systems.

B. MOTIVATION

The technical arguments for using SGI computers and IRIS Performer are somewhat mitigated by the considerable cost of acquiring high-end SGI products. Since the stated goal of the NPSNET Research Group is to promote the use, understanding and appreciation of virtual reality, platform-dependent, proprietary computers and software are not ideal. In some cases, inaccessibility, due to cost, of SGI workstations may mean that fewer interested parties will be able to learn about and contribute to the growing base of networked VR. In others, work will continue, but at a slower pace than would be otherwise possible, as competing interests share a small number of these expensive machines. In still other cases, and particularly in the case of the armed services, the lack of portability prevents potential customers from getting to use the currently available NPSNET training software because it will not run on the personal computers (PCs) which populate military desktops around the world.

The goal of this thesis is to produce an underlying layer of software designed to make it possible to port NPSNET graphics software to the lower cost, more common platform, the PC. Two approaches to this problem were taken. The first created a library of graphics

calls which simulate the syntax and functionality of Performer calls, but which have been redefined in terms of the Gemini Technology Corporation's OpenGVS™ API, which is capable of running on both SGI and Microsoft® Windows NT™ (NT) PC platforms. The primary advantage of this approach was to take advantage of existing legacy code, the result of all previous NPSNET work done on SGI workstations, when working toward a new PC version of NPSNET. The second approach concentrated on a new architecture for NPSNET software, which would provide platform-specific graphics modules which could be run with a platform-independent main controlling program.

C. SUMMARY OF CHAPTERS

Chapter II provides an overview of background considerations to this portion of the project, to include hardware, software and operating system concerns for the porting of NPSNET graphics. Chapter III details applicable porting methodology issues and discusses the implementation decisions needed to achieve the initial port of the Performer API to the OpenGVS API. Chapter IV provides the results of the project, evaluates its success and quantifies its usefulness. Chapter V introduces the second approach to the porting problem, the creation of an OpenGVS based display manager for a redesigned NPSNET architecture. Chapter VI lists conclusions gleaned from the thesis work overall, and makes recommendations for the direction of future efforts.

Appendix A is a Porting Guide to the Performer-to-OpenGVS project software. Appendix B is a complete listing of the project library source code. Appendix C is the listing of the OpenGVS programs that were ported and the original Performer programs.

Appendix D is the proposed open NPSNET architecture source code listing written in OpenGVS.

II. PERFORMER TO OPENGVS PROJECT CONSIDERATIONS

A. CHAPTER OVERVIEW

This chapter reviews the hardware and software requirements for the project to port the Performer API to the PC platform. Decisions about the selections are discussed, and quantitative comparisons of the available choices are provided.

B. TARGET PLATFORM

The primary advantage of the proprietary SGI and IRIS Performer implementation of NPSNET is, of course, its outstanding graphics performance. In addition to being an easy development platform to learn and use, IRIS Performer produces high quality computer simulations which can be run in real-time on high-end graphics workstations, a characteristic essential to the creation of a believable virtual environment.

Only recently has the PC world become a source of viable alternatives to high-end graphics-specific workstations. Advances in PC performance combined with continued attractive (and in some cases, *greatly decreasing*) pricing in a competitive market have resulted in the development of high quality, graphics-specific PC components, operating systems (OSs), and APIs.

1. Software Support

The project which became the subject of this thesis was initiated when one such emerging graphics product aimed at the PC platform, Gemini Technology Corporation's OpenGVS multi-platform graphics development software, was made available in a cooperative agreement with the NPSNET Research Group. OpenGVS, like IRIS

Performer, is easy to learn and use, and takes advantage of the unique capabilities of currently available, high end, PC three-dimensional (3D) graphics hardware. OpenGVS software facilitated a smooth programming transition from SGI workstations to PCs [GEMP96].

With a target API already selected for the project, decisions concerning the inclusion of competing OSs and hardware were greatly simplified. At the time the OS was selected, OpenGVS libraries existed only for SGI systems and PCs running the Microsoft Windows NT operating system. (OpenGVS has subsequently been made available for other OSs.) These two OSs support OpenGVS because they come with library support for SGI's OpenGL® graphics API, on which OpenGVS is based. Consequently, as the only PC OS, NT was adopted for this porting project.

Since our decision of which OS to use in this project was made, Gemini has expanded their OpenGVS support to virtually any platform for which OpenGL or Microsoft's Direct3D™ is available. This includes platforms running: IRIX™, DEC ALPHA, HP/UX, Windows NT, Windows 95®, and Microsoft's MS-DOS® operating systems. For systems that do not support either one of these APIs, it is possible to use the hardware abstraction layer (SGL) that is included within OpenGVS and is based upon OpenGL. [GEMI96]

2. Hardware Support

The configuration of the project's test PC platform was chosen based on several factors. First, the PC needed to be of the type most likely to be available to the maximum number of interested fleet consumers. This meant that despite all other issues, the target platform had to remain what end users would consider a PC. Despite the availability of

high-end graphics PCs in the price range of low-end graphics workstations, these would not be found in the fleet, and were therefore not considered. Other issues considered were system cost, graphics performance, scalability, and the availability of software to end-users.

The PC platform chosen for this project was configured with an Intel 166 MHz Pentium, the fastest Pentium-CPU in its class at the time, and 64 MB of EDO memory. Table 1 details the performance of the Intel CPUs in computationally intensive environments.

Through the use of a 3D graphics-specific video card equipped with an on-board dedicated graphics processor, performance on Intel® Pentium®-based PCs is greatly improved when compared with the performance of otherwise similarly equipped computers with standard PC video adapters. The ability of the graphics hardware to do it's considerable computational work without burdening the host CPU (which consequently slows the entire simulation system) is essential.

The high-end ELSA™ GmbH GLoria™-8 graphics board has been rated among the best performing PC accelerated 3D graphics boards available at the time of this work, and is sold with driver support for NT accelerated for OpenGL [WILL95][SCHE95]. The ELSA GLoria-8 is powered by an S3® Inc. Vision968™ accelerator for 2D graphics and a 3Dlabs® Inc. Ltd., GLINT 300SX™ for 3D graphics, and contains 8MB of Video RAM (VRAM) and 8MB of Dynamic RAM (DRAM).

Table 1: iCOMP®[a] Ratings [IDEA96]

Processor	iCOMP index 2.0	iCOMP index 1.0
Pentium - 200MHz	142	N/A
Pentium - 166MHz	127	1308
Pentium - 150MHz	114	1176
Pentium - 133MHz	111	1110
Pentium - 120MHz	100	1000
Pentium - 100MHz	90	815
Pentium - 90MHz	81	735
Pentium - 75MHz	67	610
Pentium - 66MHz	N/A	567
Pentium - 60MHz	N/A	510
Intel486™ DX4-100	N/A	435
Intel486 DX4-75	N/A	319
Intel486 DX2-66	N/A	297
Intel486 DX50	N/A	249
Intel486 DX-33	N/A	166
Intel486 SX-33	N/A	136
Intel486 DX-25	N/A	122
Intel486 SX-25	N/A	100
Intel386™ DX-33	N/A	68

a. iCOMP (Intel Comparative Microprocessor Performance) Index compares the relative performance of different Intel processors. iCOMP 2.0 has been designed solely for Intel Pentium processors. [INTE96]

The GLoria-8 video board was chosen for the project, and performance improvements from 65% to 84% were observed over the performance of similar PCs with standard 1 MB, video cards. Although GLoria-8 drivers are available for other OSs, they are not accelerated for OpenGL like the one for NT, supporting the choice of the NT OS. Table 2 shows performance comparisons between the major PC video cards considered for this project. The ELSA GLoria-8 video card is identical to the ELSA GLoria-4, but includes an additional 4MB VRAM which allows for a greater display of colors at higher resolutions, a higher refresh rate and a slight performance improvement over the ELSA GLoria-4.

C. GRAPHICS PERFORMANCE ON NPS TEST SYSTEMS

Once the intended target platform was constructed, a preliminary survey of frame rates was made using the OpenGVS API-driven software running on a variety of SGI and Intel CPU-based computers. The survey results were computed by individually displaying on each platform seven different 3D graphical models of varying geometric complexity, and averaging the application frame rates for the seven test runs. Figure 1 and Figure 2 are example images from the test program, here showing the views of the test run of an F16 model run on an NT PC platform (Figure 1), and on an SGI Indigo2 IMPACT™ (Figure 2). Although the SGI platforms in general demonstrated a slightly higher degree of anti-aliasing, the quality of the rendering was nearly equivalent on the high-end PC and SGI platforms, and the main differences in the survey runs were limited to frame rates alone. The frame rates listed in Table 3 indicate the relative performance of the various hardware configurations tested, each running the identical demonstration software depicted in Figures 1 and 2, both with and without textures.

Table 2: High-End Graphics Board Comparisons

Video Board (board/maker)	Price as of 10/96	Controller	Byte Best Overall Index[a]	Winmark 1.0[b]	Notes
ELSA GLoria-4 ELSA Inc.	$2,500	S3 Vision 968/ GLint 300SX	8.33	6.36	Accelerated Gouraud shading, antialiasing, stenciling, fogging, alpha blending, clipping, and texture mapping
3Demon™ Omnicomp™ Graphics Corp.	$1,995	S3 Vision 968/ GLint 300SX	N/A	4.77	Requires additional VGA card.
AG300™ Accel Graphics™ Inc.	$2,195	S3 Vision 968	N/A	6.36	Supports only 2 display modes. Requires additional VGA card. Accelerated alpha blending, dithering, and 16-bit Z buffering
Velocity 64 STB® Systems, Inc.	$500	S3 Vision 968	7.92	N/A	Does not support 1024X768X32K or 1024X768X16.7
Stealth64 Video 3400XL Diamond Multimedia Systems, Inc.,	$569	S3 Vision 968	7.92	N/A	Does not support 3-D acceleration

a. The Byte Best Overall Index is an index developed by Byte magazine used to compare video cards. The index compares not only the performance of the various boards at different video resolutions running common software applications, but also takes into account the features and usability of the video board. A higher index indicates higher performance. [KANE96].
b. The Winmark Index is an index developed by 3D Design magazine used to compare high-end video cards. It compares the performance of the various boards at different video resolutions running complex 3D models in both wireframe and real-time shading modes. A higher index indicates higher performance. [SCHE95]

The majority of the development work for the porting project was accomplished on SGI workstations and later tested on Intel Pentium-based PCs at the NPSNET Computer Graphics and Video laboratory. The low-end PCs listed in the survey results were used mostly for testing outside the lab.

D. CHAPTER SUMMARY

The final platform decided upon for the project to port IRIS Performer to the PC environment was a high-end PC with a 166MHz Pentium processor and an ELSA GLoria-8 video card running the Windows NT OS, chosen on the basis of price, performance, and availability to the military. Gemini Technology's OpenGVS was the graphics API used to achieve the stated goals of the project.

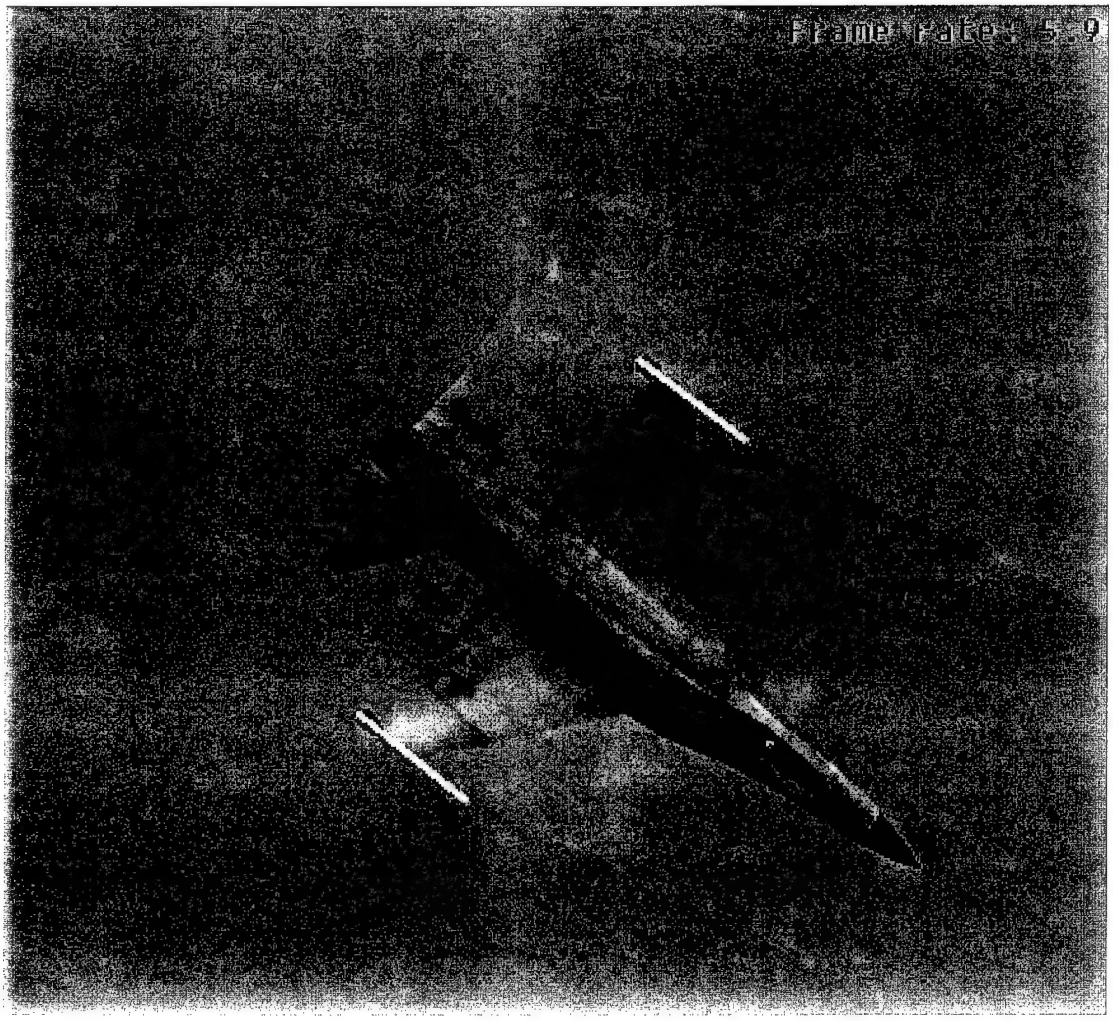


Figure 1: Example OpenGVS Test Demo on P-166

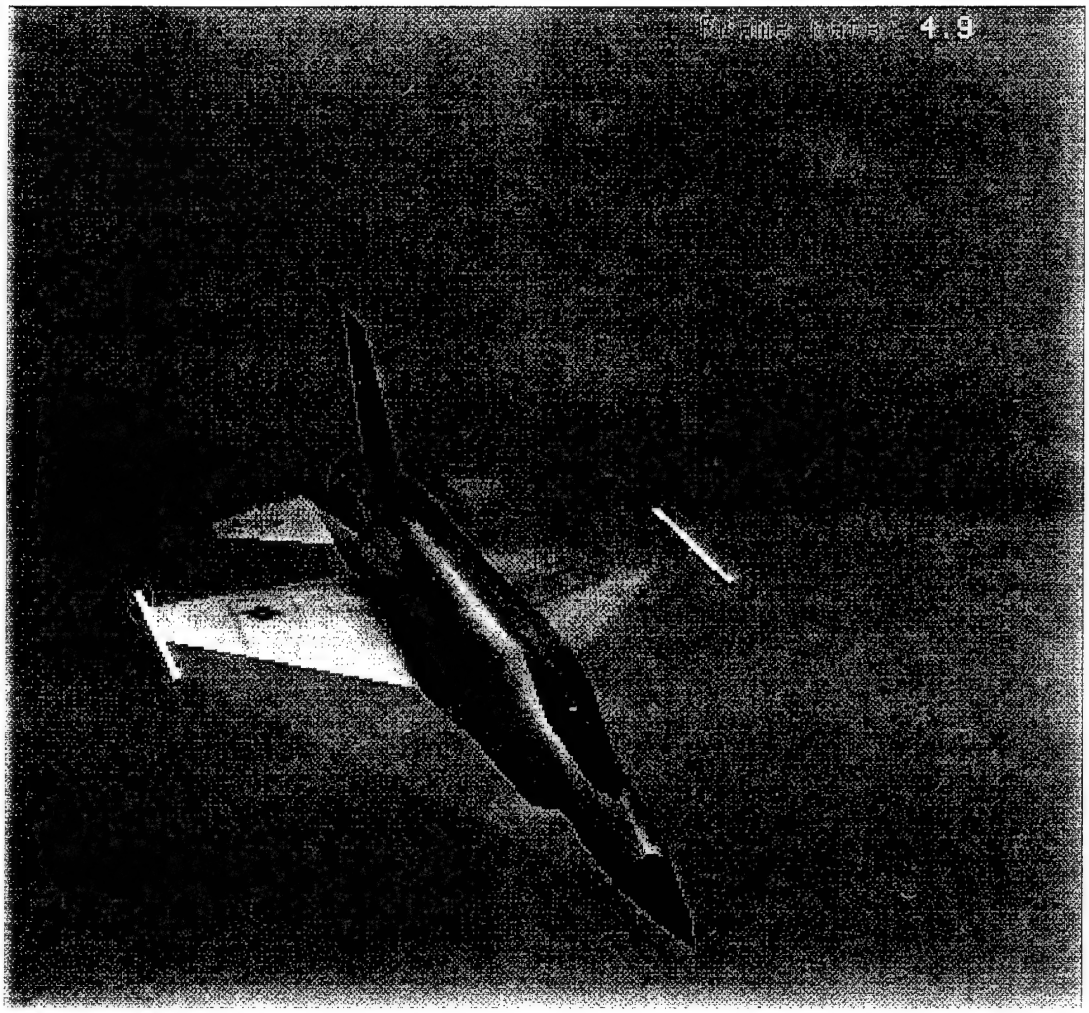


Figure 2: Example OpenGVS Test Demo on an SGI Indigo2 IMPACT

Table 3: Graphics Performance of NPS Test Systems

	32K Colors Frames/Second		True Color 16.7 Million Colors Frames/Second	
System	Textured	Non-Textured	Textured	Non-Textured
Intel486DX2 66™[a]	0.4	2.0	N/A	N/A
P5-120[b]	1.9	8.7	1.9	8.7
P5-60 GLint[c]	1.8	11.0	1.7	10.0
Indigo2 IMPACT[d]	N/A	N/A	2.3	22.0
P5-120 GLint[e]	5.7	29.4	5.0	25.9
P5-166 GLint[f]	8.4	44.6	7.3	34.9
Intergraph®[g]	N/A	N/A	56.3	68.9
Onyx™ RE-II[h]	N/A	N/A	60.0+	60.0+

a. System containing: Intel 80486DX2 66 MHz, 32MB 60ns RAM, 256K cache, and VESA Local Bus 1 MB Video Card. System cannot support true color.
b. System containing: Intel Pentium 120 MHz, 32MB 60ns RAM, 256K cache, and PCI 1 MB video card.
c. System containing: Intel Pentium 60 MHz, 40MB 60ns RAM, 256K cache, PCI ELSA GLoria 8.
d. Indigo2 IMPACT system containing: 150 MHz MIPS R4400™, 64MB main memory, GUI Extreme Graphics Pipe.
e. System containing: Intel Pentium 120 MHz, 32MB 60ns RAM, 256K cache, and PCI ELSA GLoria 8 video card.
f. System containing: Intel Pentium 166 MHz, 64MB EDO 60ns RAM, 512K pipeline burst cache, and PCI ELSA GLoria 8 video card.
g. Intergraph Corp. TDZ-40, Dual Intel Pentium 100 MHz, 128MB 60ns RAM, 512K cache, and Intergraph accelerated video card. System supports only true color.
h. Onyx RE-II system containing: Four 150 MHz MIPS R4400, Reality Engine^{2}™ Graphics Pipeline, 192 MB main memory. System supports only true color. Results listed do not reflect load limit of the platform/hardware; even higher trivial test loads did not reduce frame rates.

III. PERFORMER TO OPENGVS PROJECT IMPLEMENTATION

A. CHAPTER OVERVIEW

This chapter begins by reviewing the rationale for porting the Performer API to the PC platform. The remainder of the chapter documents the techniques used to implement Performer functionality in terms of the OpenGVS API, and the characteristics of the two APIs which made those methods necessary.

B. PORTING METHODOLOGY CHOICES

The most direct method for porting NPSNET graphics to the PC platform would have been to simply rewrite all IRIS Performer specific code in equivalently functioning OpenGVS code by hand. The resulting software would efficiently take advantage of the many features of OpenGVS without any unnecessary overhead. This method would theoretically bring about the highest run-time performance for OpenGVS driven graphics.

This method was not initially chosen however, for the following reasons. First, this method is the least flexible. Changes made to existing NPSNET Performer code would then need to be made separately in the OpenGVS version using the OpenGVS API, not IRIS Performer, complicating maintenance. Second, NPSNET programmers would require training in both APIs in order to work on the project, decreasing productivity toward new, better versions of the software in deference to merely staying at the same level on a PC platform, and in any case detracting from the main focus of NPSNET, which is networked virtual environments, not graphics APIs. Finally, complete recoding would result in two

versions of NPSNET, each with its own support requirements for the staff to handle with NPSNET users.

The methodology adopted instead was the creation of a library of graphics functions which use prototypes identical to those in the IRIS Performer API, but which have functionality defined in terms of the multi-platform OpenGVS API. This plan had several readily apparent advantages. First, Performer-based NPSNET graphics code essentially would not need to be changed to work on the PC architecture. Since the main purpose, in general, of porting rather than rewriting code for a new platform is to take advantage of successful legacy work, all previous NPSNET work could be preserved for the PC version. Second, since only one version of NPSNET code would be needed, support and upkeep would be simplified. This version would include only IRIS Performer API function calls, with no OpenGVS calls. The PC version of the executable program would simply be a recompilation of the SGI workstation version, using the OpenGVS libraries instead of Performer's. Third, by linking to PC platform libraries, NPSNET development could move outside the Research Group's SGI lab, where previously all work needed to be done, to the PCs of the Group's programmers.

Figure 3 illustrates the use of one set of NPSNET graphics code, which could be used with either the original Performer libraries or the Performer to OpenGVS (PTG) libraries, described above. By recompiling graphics applications with the appropriate libraries, Performer code could be multi-platform.

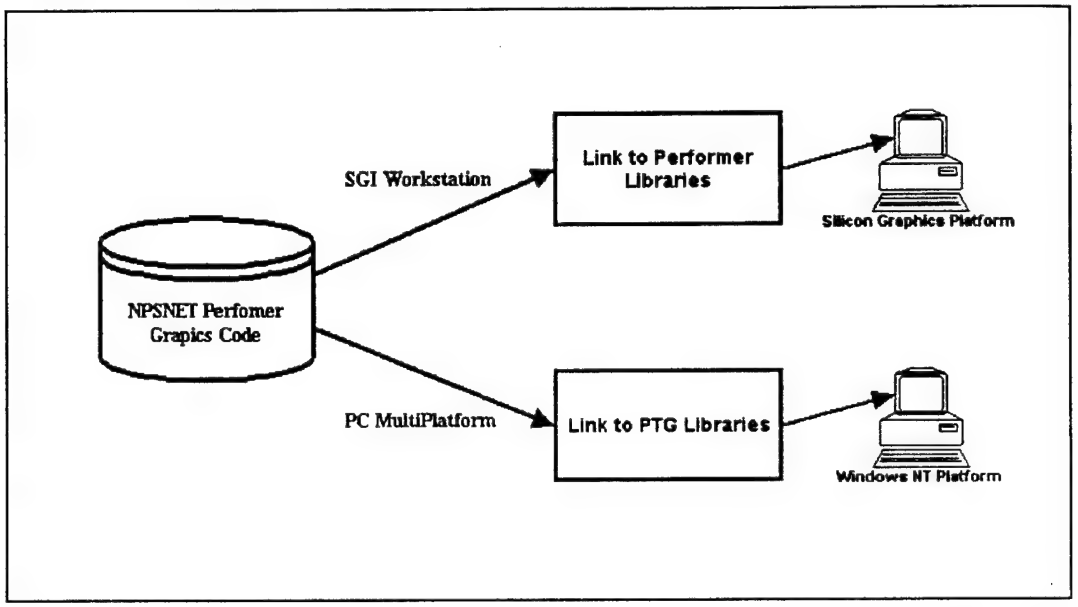


Figure 3: Platform Independent Graphics Code

Figure 4 shows a simple code fragment from the PTG libraries which illustrates how the general principle of PTG is actually implemented. The Performer function **`pfChanView`** is used to assign position (in terms of world coordinates along the x, y, and z axes) and attitude (in terms of Euler angles: heading, pitch, and roll) arguments to the viewpoint into a graphics channel. The PTG library performs the following tasks in order to get the OpenGVS graphics engine to understand a call to **`pfChanView`** (line numbers apply to Figure 4):

1. Convert world coordinate vector type from Performer 3-float array (**`pfVec3 xyz`**) to OpenGVS 3-float **`struct`** (**`G_Position position`**) lines 24-26.

```
1  /* pfChanView -
2   *    This function sets viewpoint and direction for an input
3   *    pfChannel, in terms of OpenGVS resources.
4   *    pfChannel input is converted to a GV_Channel, with a
5   *    GV_Camera attached and used to set the channel's viewpoint.
6   *    This funtion's source is in ptgLib.c .
7   */
8  void  pfChanView( pfChannel * ch, pfVec3 xyz, pfVec3 hpr )
9  {
10    /* create OpenGVS position and rotation vectors */
11    G_Position  position;
12    G_Rotation  rotation;
13
14    /* create a temporary OpenGVS camera handle */
15    GV_Camera camera;
16
17    /* inquire the current cammera in use by the input channel */
18    GV_chn_inq_camera( *ch, &camera );
19
20    /* Performer labels positive x axis as east, positive y axis as
21     * north, and positive z axis as altitude. OpenGVS recognizes
22     * these axes as x, -z, and y, respectively
23     */
24    position.x =  xyz[0];
25    position.y =  xyz[2];
26    position.z = -xyz[1];
27
28    /* convert from degrees (Performer) to radians (OpenGVS). */
29    rotation.x = hpr[1] * G_DEG_TO_RAD;
30    rotation.y = hpr[0] * G_DEG_TO_RAD;
31    rotation.z = hpr[2] * G_DEG_TO_RAD;
32
33    /* set the current camera/channel to a viewpoint with the
34     * converted position and rotatation vectors
35     */
36    GV_cam_set_position(camera,PTG_GL_currentPlatform,&position);
37    GV_cam_set_rotation(camera,PTG_GL_currentPlatform,&rotation);
38 }
```

Figure 4: Sample Performer Code Fragment

2. Convert graphics channel frustum vector type from Performer 3-float array (`pfVec3 hpr`) to OpenGVS 3-float `struct (G_Rotation rotation)` lines 29-31.

3. Perform unit conversions (i.e. Performer angles are given in degrees, OpenGVS uses radians.) lines 29-31.

4. Perform axis orientation conversions so that movement in OpenGVS will be along the same relative world axes as in the original Performer project (i.e. Performer labels positive x axis as east, positive y axis as north, and positive z axis as altitude. OpenGVS recognizes these axes as x, -z, and y, respectively.) lines 24-31.

5. Associate Performer frustum parameter arguments with the proper OpenGVS viewpoint paradigm, the `GV_Camera` which is assigned to an OpenGVS graphics channel, the `GV_Channel`, as opposed to the direct (albeit less functional) IRIS performer method of assigning the viewpoint to the `pfChannel`, lines 36-37.

Although this sample function may be misleading because it appears to demonstrate a straight-forward correspondence between the Performer and OpenGVS APIs (which does not exist in reality), it is intended to serve merely as an example of the type of OpenGVS function "wrappers" constructed to port the Performer code. As will be discussed, the bulk of the Performer functions could not be ported as easily.

There were, of course, many disadvantages foreseen with this porting method. First, rewriting a complete library of all 906 IRIS Performer 1.2 functions would be very time consuming. However, since most go unused in NPSNET, the job of porting only the most often used functions appeared achievable in the time allotted for this thesis. Second, since the PTG libraries would essentially be redirections of OpenGVS functions, there would be added computational overhead beyond that which OpenGVS normally requires. This was judged to be insignificant, however, since the majority of this overhead would be seen during simulation setup, as the graphics engine is started and resources are allocated (as

will be shown in the next section), and not during the simulation itself. In any event, the OpenGVS NT version of the software on the PC platform would never be able to compete with the performance which NPSNET previously had exhibited with the SGI version, due to SGI workstation hardware advantages previously discussed in Chapter II.

C. PORTING TECHNIQUES

The Performer and OpenGVS APIs and graphics engines organize many tasks entirely differently from each other. In several cases, major design decisions for the implementation of the PTG libraries revolved around solving compatibility problems between the two APIs, and not merely converting units of measurement, as was the case exemplified in Figure 4. The remainder of this chapter explains the implementation techniques behind some of these design decisions, as well as the characteristics of the Performer and OpenGVS APIs which precipitated them.

1. Main Program Organization

The goal of the OpenGVS API is graphics code which is portable between both NT and SGI's IRIX OS. Consequently, compilation of OpenGVS programs must be capable of producing either of two "main" functions depending upon which OS is being used for a particular compilation. This is because Windows NT programs require the use of the `WINAPI WinMain()` function as the main function, which automatically performs some window management, while IRIX main functions look more like the generic `int main(int argc, char *argv[])`, which requires programmers to handle their own

window management (or at least use other SGI platform-specific calls to do that work for them).

This issue of the main function is the first place the PTG project had to make a change in the normal structure of typical Performer programs. The typical Performer project's **`int main()`** which called it's own window managing functions was replaced by the main file Gemini uses for many of its demo applications, **`gv_main.c`**.

The original Performer code from **`int main()`** is still used in projects with the PTG libraries, but it is converted to a function **`int pfMain()`** as outlined in the Porting Guide, Appendix A, so that the remainder of the application functionality which existed in the original **`int main()`** is preserved. The **`pfMain()`** function is called later by the PTG project system after the OpenGVS graphics engine is started. Figure 5 below depicts the flow of the compiled executable under both the IRIX and NT platforms.

On the NT platform, the Microsoft Visual C++® compiler compiles the file containing the main function and adds **`WINAPI WinMain()`**, for window management, to produce Windows NT applications. Through the use the OpenGVS batch file (**`login.bat`**), compiler options are set, transparently to the user, to create an application with the necessary windowing functions.

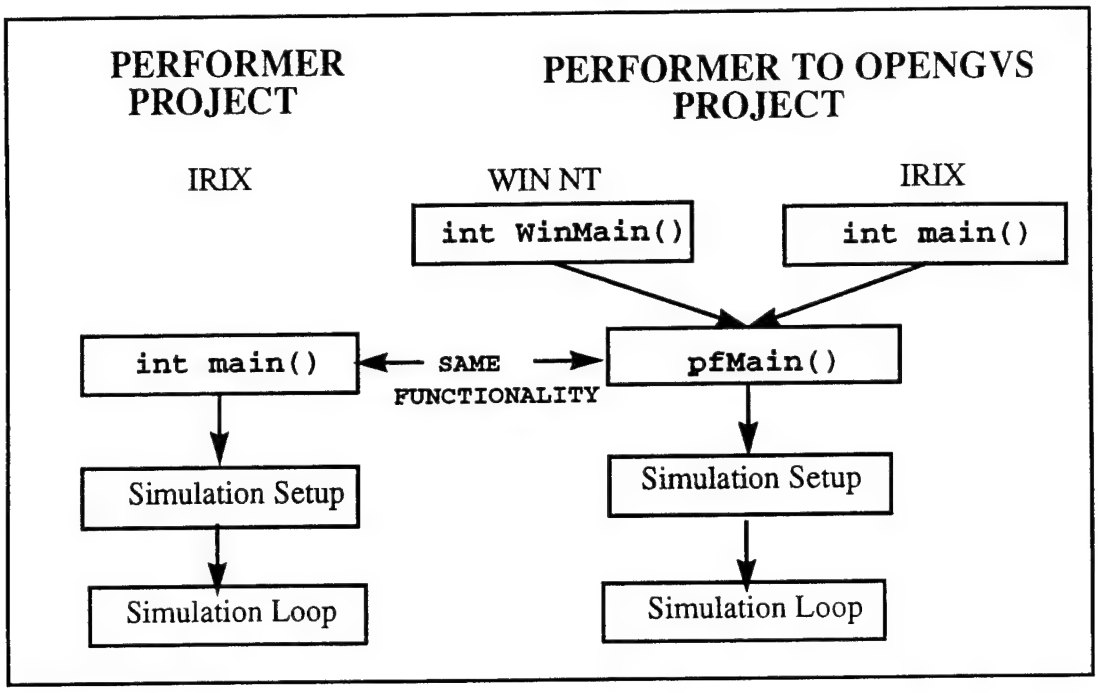


Figure 5: Example of a Graphics Application Setup

2. Graphics Resource Allocation

a. Building applications with Performer and OpenGVS APIs

Projects written with the Performer API follow a set series of steps, as listed below in Figure 6 (abbreviated for simplicity).

Ordinarily, OpenGVS projects would follow the same basic steps, but with the following minor differences. First, at the start of the OpenGVS application, callbacks are established for declaring all required resources and connecting the graphics pipeline (`GV_user_init`), and for the simulation loop (`GV_user_proc`). After the callbacks have been established, the graphics engine is started, and the callbacks evoked. Because of its role in the OpenGVS project structure, the `pfMain()` function from the original

Performer project must be incorporated into the `GV_user_init` function in PTG projects.

1. Establish variables for the required elements (e.g. `pfScene*, pfPipe*, pfChannel*`)
2. Initialize the IRIS Performer engine (e.g. `pfInit();`)
3. Load or create the database (e.g. `root=LoadFile(argv[1],NULL);`)
4. Specify (create) the scene the channel should draw (e.g. `scene = pfNewScene();`)
5. Add the root of the database to the scene (e.g. `pfAddChild(scene, root);`)
6. Initialize the graphics pipeline (e.g. `p = pfGetPipe(0); pfInitPipe(p, OpenPipeline);`)
7. Create a channel on the pipe (e.g. `chan = pfNewChan(p);`)
8. Configure the channel (e.g. `pfChanScene(chan,scene);`)
9. Repeatedly render the scene using a simulation loop (e.g. `pfFrame();`)

Figure 6: Steps Involved in Building a Performer Application From Ref. [HART94]

b. Allocating Resources

This new use for what was once a main function creates a problem for PTG projects caused by the inabilities to either predict the number and type of resources that will be needed in the portion of `GV_user_init` made up by `pfMain()` (i.e. variable declaration), or to be able to refer to them later due to scope and lifetime problems. To solve these problems with as little awareness as possible required by the Performer programer, the PTG libraries use the function `PTG_GlobalInit` to create and initialize a cache of resources. It is from this cache that the Performer programmer seamlessly draws. These resources will already have been declared/created in the proper sequence for OpenGVS by

the time the original Performer code would need them, and they may be referenced at any time via handle by the PTG system.

The constructs which perform these tasks can be seen in **pfToGVS.h**, **ptgUtils.c**, and **ptgLib.c**. They are exemplified by the case of the resource **pfScene**, the most straightforward of the resources to port, as condensed in Figure 7.

```
1  /* Equate types (excerpt from pfToGVS.h) */
2  typedef GV_Scene     pfScene;
3
4  /* For cache of resources (from pfToGVS.h) */
5  #define      AVAIL_SCNS    8L
6  long         PTG_GL_allocatedScns;
7  GV_Scene     PTG_GL_scn[AVAIL_SCNS];
8
9  /* Init cache to null (from pfUtils.h) */
10 PTG_GL_allocatedScns = 0L;
11 for (ix=0; ix<AVAIL_SCNS; ix++)
12 PTG_GL_scn[ix]=0;
13
14 /* Allocate resources for Performer project
15  *using cached PTG resources (from ptgLib.c)
16  */
17 pfScene * pfNewScene( void )
18 {
19    if ( PTG_GL_allocatedScns >= AVAIL_SCNS )
20    {
21       printf( "PTG: Fail pfNewScene. Not
22         enough AVAIL_SCNS.\n" );
23       printf( "PTG: Usable pfScenes 0 - %d.
24         Exiting PTG.\n", AVAIL_SCNS - 1 );
25       exit( G_FAILURE );
26    }
27   GV_scn_create( &PTG_GL_scn[ PTG_GL_allocatedScns ] );
28   return &PTG_GL_scn[ PTG_GL_allocatedScns++ ];
29 }
```

Figure 7: Example Resource Allocation

3. Handling Performer's Scene Graph

In order to create a scene to view in Performer applications, a scene graph must be built and the scene associated with a channel. This is accomplished with a call to **`pfChanScene`**.

Scene graphs are made of connected database units called nodes. Nodes, as can be seen in Figure 8, can be roots, branches or leaves. This scene hierarchy enables logical traversal for efficient cloning, culling, deleting, drawing, flattening, intersecting, and state inheritance [HART94]. An action which is performed on a scene graph node applies to all children of that node as well. Since the **`pfScene`** is the graph node linked to the channel for drawing, it is always the root node (having no parents).

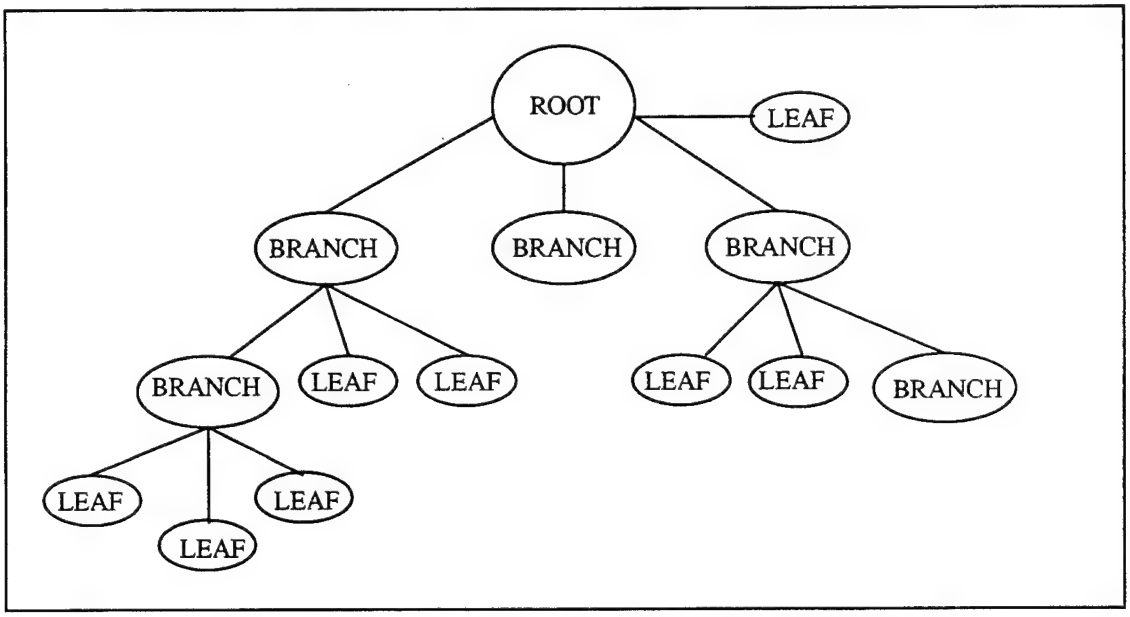


Figure 8: Example Hierarchy From Ref. [HART94]

Another type of hierarchy between potential scene graph nodes, derivation hierarchy, is depicted in Figure 9. All other elements types which can be added to the scene graph as nodes, such as **pfSCSs**, **pfDCSs**, **pfGroups**, and **pfNodes**, inherit attributes from the **pfNode** class.

Once a root is established, additional nodes can be added to the scene, as in our previous example, Figure 6, in which a file (imported as a **pfNode**) was added as a child to a **pfScene**. The function **pfAddChild(pfGroup*,pfNode*)** adds child nodes to parent nodes to expand the scene graph. Arguments to the **pfAddChild** function can be any type inheriting attributes from the **pfGroup** and **pfNode** classes (except that **pfScenes** are never children in scene graphs).

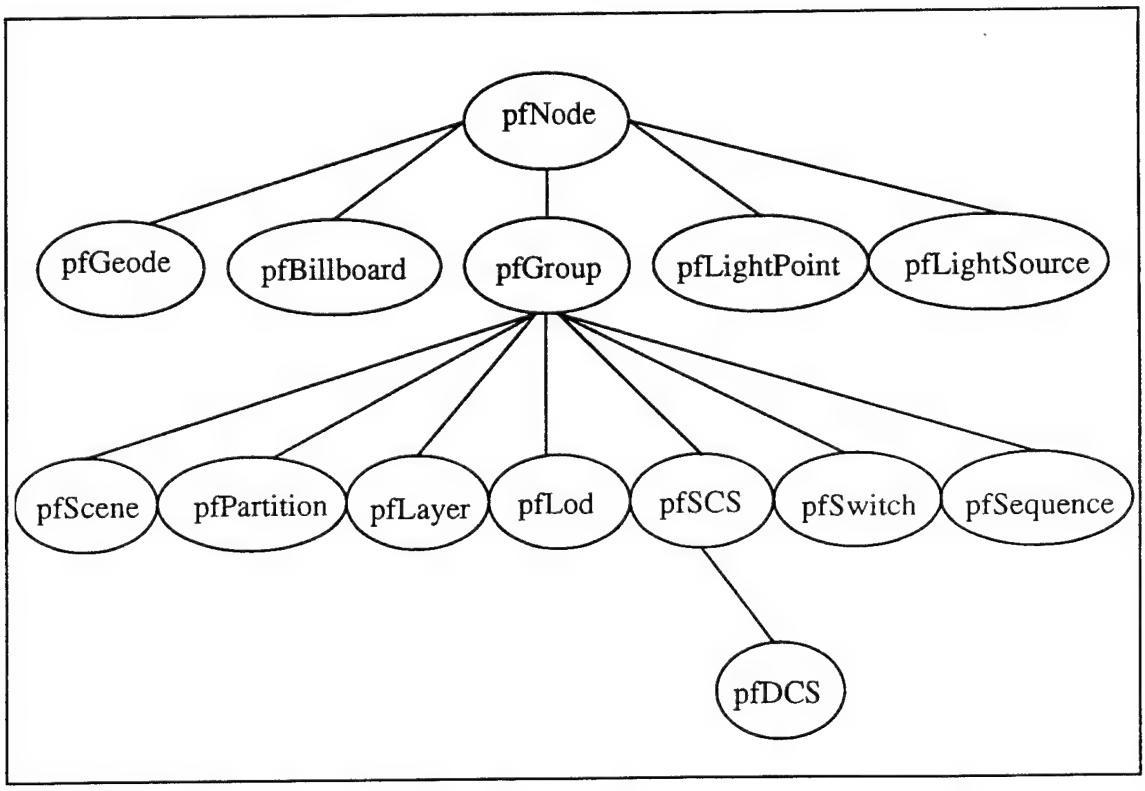


Figure 9: Type Hierarchy for the libpf Node Types From Ref. [HART94]

This Performer API framework for application scene building is easy to learn and use, but it presented a problem for PTG. OpenGVS has a similar scene graph building phase in its applications, but instead of just one function to add any type of node to any other type of node in the scene hierarchy, it uses specific functions for specific cases. For example, where Performer might use **pfAddChild** to either add lights to a scene, or add objects to a scene, or attach sub-objects to parent objects (e.g. moveable control surfaces on an airplane), OpenGVS requires separate calls to **GV_scn_add_light,** or **GV_scn_add_object,** or **GV_obi_attach_child**, respectively. This is because OpenGVS objects do not all inherit from the same base class, as Performer nodes do.

To ensure that the correct OpenGVS calls are invoked from the PTG wrapper for **pfAddChild**, PTG accepts its arguments as type **void*** as in:

```
long pfAddChild(void * parentVoid, void * childVoid);
```

PTG then performs a table lookup to determine the types to which these pointers were originally allocated during **PTG_GlobalInit**:

```
parentType = PTG_resource_pointer_lookup( parentVoid );
```

The **void*** arguments are then cast to the appropriate OpenGVS function argument types, and the proper functions are invoked:

```
GV_scn_add_object(*((GV_Scene*)parentVoid),*((GV_Obi*)childVoid));
```

While this technique will be slower than the Performer original or the manner in which OpenGVS calls and objects were intended to be used, it solves a considerable problem caused by incompatible resource types between the two APIs during application setup, and should not affect runtime performance.

4. Putting Viewable Objects into a Simulation Scene

As shown in Figure 6 (step 3), Performer scene graphs can be built by importing files as nodes and then adding them, as children, onto other nodes in the graph. In many cases, these nodes are created by first "newing" them with **pfNewDCS**, or **pfNewGroup**, and then associating objects in the form of imported graphics binary files.

The closest construct to **pfDCS**s and **pfGroup**s (as well as several other basic node types) which exists in OpenGVS is the **GV_Obi**, or object *instance*. Once they have been created, **GV_Obi**s can be added to the scene graph with **GV_scn_add_object** as was shown above. However, the OpenGVS technique for creating them differs from Performer's in that they are never "newed", but merely associated with previously defined **GV_Obd**s, or object *definitions*. Viewable objects, such as can be produced from imported truck or airplane image files, are loaded as definitions, **GV_Obd**s, and with **GV_obi_instance()**, instances are created from definitions.

A two step process was used in PTG to make Performer and OpenGVS appear to work identically. First, when image files are loaded, they are associated with **GV_Obi**s in the same PTG function, so that they are immediately available to be added to the OpenGVS scene graph as nodes. Second, when new **pfDCS**s and **pfGroup**s are created, they are associated with a PTG device called a **PTG_GL_null_obd**, which is a very small OpenGL object (less than one unit in size) so that they can apparently be added to an OpenGVS scene graph the way that Performer might, prior to having been associated with a viewable object definition. If the Performer application later calls for the new instance to be associated with an object image file node with **pfAddChild**, the OpenGVS function

which associates two **GV_Obis**, **GV_obi_attach_child**, is used to build that section of the scene graph.

Although this method appears convoluted at first glance, it allows the flow of Performer application calls to be unchanged for the sake of use with PTG. Further, the expense, in terms of processing time spent on this object organization, is incurred during **GV_user_init**, and not during the simulation loop, where it would slow the frame rate on a PC.

5. Graphics Channels

As explained above, Performer programmers build graphics pipelines (with **pfGetPipe**) and then create channels on those pipes (with **pfNewChan**) which are assigned to specific scenes (with **pfChanScene**). Any number of channels can be added to a pipeline, each with its own viewport, or sub-window, which is displayed inside the main window (pipeline), like a picture inside a larger picture. The area allocated to each sub-window may be specified using **pfChanViewport**. Figure 10 depicts an IRIS Performer demonstration program, **multichan**, which illustrates the use of four channels (all showing the same scene in this case) inside a single pipeline, each assigned to one quarter of the window.

The OpenGVS equivalent to the **pfPipe**, the **GV_Fbf**, also may contain many sub channels. The only difference in implementation is that when multiple channels are being displayed in a single **GV_Fbf** window, they must be attached as sub-windows inside a parent channel, which in turn is attached to the **GV_Fbf**. Only one parent channel may be attached to a **GV_Fbf**.

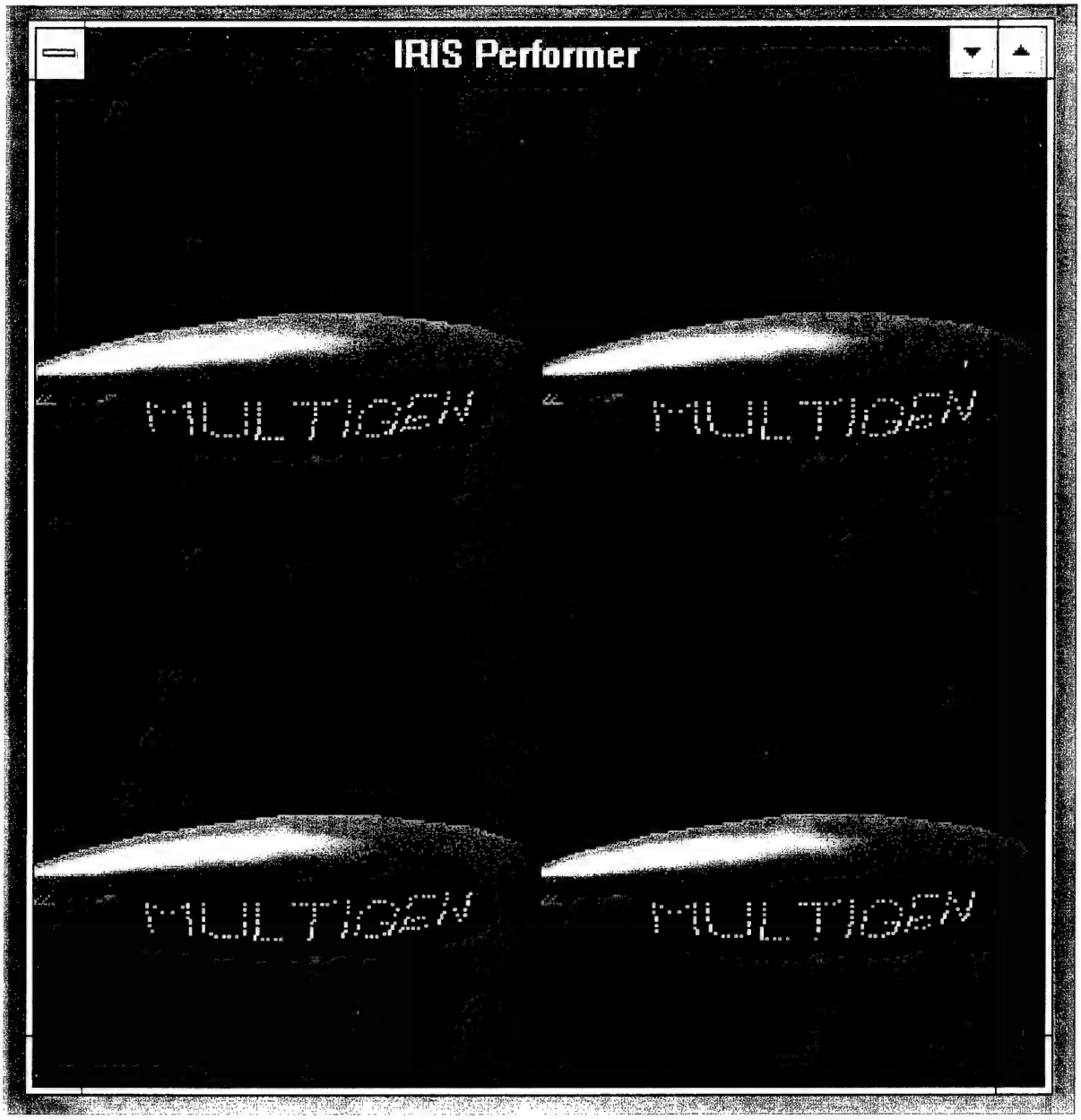


Figure 10: PTG Multichan Demo

The resulting implementation for PTG, therefore, requires that for each **GV_Fbf** used in the original Performer application, a parent channel must be attached automatically. That way, whether only one sub-channel is attached to the pipeline or many, they will never be attached directly, but always as children of the **GV_Fbf**'s parent channel.

This construct is necessary because PTG never knows beforehand how many channels will be added to a window. If it assumed only one, as exists in most simulations, an attempt by a programmer to add a second would fail. To ensure the safety of PTG, in general, Performer programs with *n* pfChannels will always result in PTG ports with *n+1* GV_Channels.

6. Programming Visual Environmental Effects

Porting visual environmental effects (e.g. **pfEarthSky, pfFog, pfSmoke, and pfLight**) from Performer to OpenGVS required special resource allocation devices, similar to those already seen in PTG. Even then, however, due Performer's underlying hardware requirements, some environmental effects could not be identically ported to a non-SGI system. In cases when some functionality could not be duplicated in the PTG libraries, the PTG software alerts the programmer, and uses other pre-defined defaults to allow the project to compile and run satisfactorily.

a. pfEarthsky Effects

Performer **pfEarthsky** visual effects have corresponding functions in OpenGVS, the Infinite Sky/Ground utilities, **GVU_isg**. However, in OpenGVS, the **GVU_isg** functions are designed to directly affect channels instead of the separate

pfEarthsky models which Performer configures separately and then adds to its channels. A comparison of the steps in configuring environmental effects using Performer and OpenGVS is shown in Figure 11.

Performer Environmental Effects Setup

1. Create the Performer earthsky model:

```
pfEarthSky * esky;
esky = pfESky();
```

2. Specify the mode, attributes, and color of the model:

```
pfESkyMode(esky,PFES_BUFFER_CLEAR, PFES_SKY_GRND);
pfESkyAttr(esky,PFES_GRND_HT,-1.0f * bsphere.radius);
pfESkyColor(esky,PFES_GRND_FAR,0.3f,0.1f,0.0f, 1.0f);
pfESkyColor(esky,PFES_GRND_NEAR,0.5f,0.3f,0.1f,
    1.0f);
```

3. Finally, associate the newly created model (**esky**) with an existing channel (**chan**).

```
pfChanESky(chan, esky);
```

OpenGVS Environmental Effects Setup

1. Setup the channel:

```
GV_Light sun;
GVU_isg_setup( channel );
```

2. Add earthsky functionality:

```
GVU_isg_set_light ( channel, sun );
```

Figure 11: Creating a Simple EarthSky Mode in Performer and OpenGVS

The **pfEarthsky** model is configured with attribute values prior to actually associating the model to a channel. Conversely, OpenGVS requires that a channel be

configured for infinite sky/ground graphics *prior* to any other changes being made to the original configuration of the channel, such as modifications to sky or land color.

Because it is the environmental model and not the channel that is being modified in Performer, a data structure for storing intended **pfEarthsky** attributes had to be created for use in the PTG project so that the port would be able to handle cases in which a channel had not been identified prior to creating a **pfEarthsky** model. An array of **pfEarthSky** data structures was therefore created during the initial setup of the PTG project in **pfToGVS.h.** This approach is similar to the creation of scenes that was described in the previous section. This device allows the initialization of a GVS channel with the temporary earthsky data structure when **pfChanESky** is called. Once an association is made between the **pfEarthSky** model and the channel with **pfChanESky**, all previously stored changes (as well as subsequent modifications) to the **pfEarthSky** model will result in immediate changes to the channel in the PTG program.

An additional, and to the viewer a more noticeable, difference in the way Performer and OpenGVS handle environmental effects, is the ability of OpenGVS to automatically blend and shade land and ground colors based upon scene lighting. Performer requires the programmer to assign two near and far colors for sky and ground models, while OpenGVS only sets one color each for the sky and ground planes, which it then modifies using the scene's lighting model to create varying shades throughout the scene. With that in mind, the PTG implementation always allows OpenGVS to control the shading of the scene, and needs only one of the two colors for the sky and ground planes.

b. Fog Effects

Fog in Performer and OpenGVS is created in roughly the same manner, using distinct fog models. However, while Performer allows the programmer to actually *apply* a fog in an application (not merely *create* one) without first defining a channel or a scene, OpenGVS requires the fog model be first created and then applied directly to either a channel (**GV_chn_set_fog(channel,fog_model)**) or a scene (**GV_scn_add_fog(scene,fog_model)**). Since there may be a case in which a Performer programmer would create and then apply a fog model prior to creating a channel, a fog array, similar to the scene and earth/sky arrays, is used in PTG to store the fog model, which is then accessed and applied only after a channel has been associated with the fog. Therefore, if in **pfMain**, **pfApplyFog** is called prior to the definition of a channel, the PTG port will wait and apply the fog to the active channels in the function **PTG_post_pfMain**. Alternately, if the channel has already been declared when a Performer application applies a fog model, then the fog is immediately applied to the channel in **pfApplyFog**.

There are seven different types of fog available in Performer, only three of which are used extensively in most programs. These three models calculate their appearance based upon the vertices (**PFFOG_VEX**) of the bounding areas of the fog. The remaining four types utilize more time consuming calculations which construct each individual pixel of the fog model (**PFFOG_PIX**). The three vertex fog types (**PFFOG_VTX_LIN**, **PFFOG_VTX_EXP**, and **PFFOG_VTX_EXP2**) correspond directly to OpenGVS's fog

functions (**GV_FOG_TYPE_LINEAR**, **GV_FOG_TYPE_EXP**, and **GV_FOG_TYPE_EXP**).

The pixel fog modes are not used in most Performer programs because they are both too time intensive and not supported by some of the lower-end SGI systems (i.e. IRIS Entry, Indy, XL, XS, XS24, XZ, Elan, Extreme, and VGX)[HART94]. OpenGVS does not support pixel fog calculations, but in order to make the project handle as many Performer demos as possible, the PTG port maps the pixel modes to its associated vertex mode automatically (e.g. **PFFOG_PIX_LIN = PFFOG_VEX_LIN**). The most time intensive fog model, **PFFOG_PIX_SPLINE**, only effective on Reality Engine graphics systems, has been mapped to **GV_FOG_TYPE_LINEAR** to allow PTG projects to compile and run as efficiently as possible.

c. Smoke Effects

Although Performer and OpenGVS have similar smoke functions, the techniques for creating these effects are vastly different. Performer creates smoke by texture mapping a simple 2-dimensional plane, then modifying the plane as time varies (**pfuNewSmoke**), while OpenGVS creates a three-dimensional smoke model definition that is manipulated like any other OpenGVS object.

Performer creates fire the same way it does smoke. OpenGVS, however, creates fire differently from smoke, not as a model definition, but rather as an animation definition (**GVU_anim_create**). However, to model more closely to the Performer version, PTG/ OpenGVS fire is implemented by changing the color and state of the smoke during the

lifetime of the smoke. The fire starts out as a bright orange-red "smoke", rises to a thick black smoke, then finally dissipates as time increases, as it appears in Performer.

D. CHAPTER SUMMARY

Although there are differences in the structure of the Performer and OpenGVS APIs, the fact that they follow similar steps in creating their simulations made redefinition of Performer functionality in terms of OpenGVS possible. In general for every program, resources are allocated, scene graphs and graphics pipelines are constructed, and simulation loops are used to update and redraw the scene. The majority of the porting work, therefore, involved caching resource creation calls to ensure adequate scope and lifetime of objects in the ported applications.

IV. PERFORMER TO OPENGVS PROJECT RESULTS

A. CHAPTER OVERVIEW

This chapter summarizes the PTG project results. The usefulness of the library is quantified both in terms of porting SGI specific code to the PC platform in general, and as a tool for users of NPSNET.

B. PROJECT RESULTS

The effort to build a library of wrappers for the SGI proprietary IRIS Performer API using the multi-platform OpenGVS API met with mixed success for NPSNET. The actual coding and testing of the PTG library was tedious but straight-forward. The library itself produced adequate functionality with most of the major graphics capabilities of Performer being implemented in OpenGVS. Validation of this effort was accomplished by recompiling seven Performer sample programs on PCs using the PTG library. The PC versions of these demonstration programs, which are provided by SGI with the Performer API to illustrate its most commonly used features, look like, and function like, their Performer equivalents.

The features of Performer implemented in the PTG library include type handling (e.g. **`pfPipe`**, **`pfChannel`**, **`pfNode`**), resource allocation (e.g. **`pfNewScene`**), graphics pipeline construction (e.g. **`pfNewChan`**, **`pfChanScene`**), scene graph construction (e.g. **`pfAddChild`**, **`LoadFile`**), frustum manipulation (e.g. **`pfChanFOV`**, **`pfChanView`**), environmental effects (e.g. **`pfChanESky`**, **`pfApplyFog`**), utility functions (e.g. **`pfSetVec3`**, **`pfGetNodeBSphere`**), as well as object configuration functions (e.g.

pfFogColor, **pfESkyMode**). In all, support for 177 IRIS Performer functions found in the demo programs (as well as in NPSNET itself) were included in the PTG library.

The PTG Porting Guide, Appendix A to this thesis, details the steps required to create NT programs from IRIS Performer source code and the PTG library contained in Appendix B, as well as the steps to use the OpenGVS versions of the sample programs which are provided in Appendix C.

C. PTG IMPLICATIONS FOR NPSNET

Despite the capability to compile graphics software written with the Performer API on the NT platform using the PTG library, NT's use specifically for NPSNET software as a whole is considerably more problematic. The current architecture of NPSNET prevents the separation of the portable code from the other SGI IRIX dependent code. IRIS GL functions, IRIX (and generic unix) OS system functions, networking functions, user interface functions, and other platform-dependent categories of software, intermixed with the Performer graphics code, prevent portability for the entire NPSNET system, at least without libraries of wrapper functions to convert other non-graphics, SGI-specific software for use with NT platform. The OpenGVS library makes the graphics code portable, but has no functionality for these other categories.

Therefore, the initial goal of accomplishing even a stand-alone (not networked) NPSNET simulation operating on the PC platform, by using the PTG libraries alone to make the SGI code portable, was not possible. The creation of libraries to convert the remaining platform specific code to the PC/NT platform is beyond the scope of this effort, and in any case, may not be the best way to solve the problem. The addition of new libraries

would not prevent revisiting the NPSNET architecture issues faced in this work in the event that a new, and currently unforeseen effort to port NPSNET to a different platform (or to a different graphics engine) was initiated.

V. OPEN NPSNET ARCHITECTURE

A. CHAPTER OVERVIEW

The preceding chapters described the work of porting SGI workstation code to an NT-based API. Issues outlined in Chapter IV suggest there may be better methods for producing a multi-platform version of NPSNET. This chapter introduces a proposed new architecture for NPSNET, designed with modularized functional and platform-dependent components, which may alleviate the disadvantages of the PTG system.

B. NPSNET OPEN ARCHITECTURE DESIGN

The PTG project produced two valuable findings for future work toward a multi-platform version of NPSNET. First, PCs and the tools used in PTG (ELSA GLoria graphics hardware, NT OS, Visual C++ compiler, OpenGVS graphics engine) are capable of producing quality graphics applications for PCs, validating an initial assumption about platform suitability. Second, a new object oriented architecture for NPSNET, designed to modularize application components into functional and platform-specific subcomponents, could greatly facilitate the effort.

Independent of the PTG work, the NPSNET Research Group has been in work on the design phase of a new architecture for NPSNET. This architecture, called Open NPSNET, although not yet finalized, incorporates the design depicted in Figure 12 below.[BARK96]

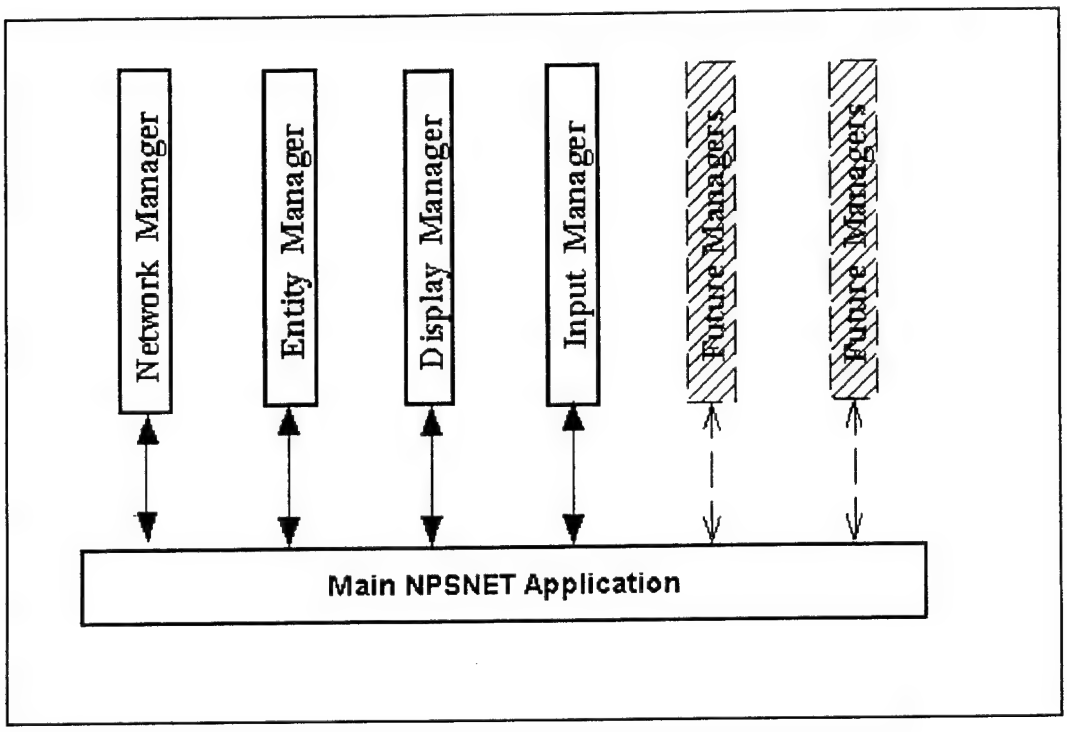


Figure 12: Proposed NPSNET Architecture [BARK96]

In Open NPSNET, the main program can be viewed as a bus into which functional components are plugged. These components, called managers, are envisioned to be designed to individually, and independently, handle the details of networking, user input, entity bookkeeping, display control, and the like. These managers send and receive messages between each other via a conduit controlled by the main program, which may include shared memory, interprocess communications, parallel computing networks, or some other globally accessible means.

The key multi-platform characteristic of the new architecture, however, is the object oriented design of the main components. If the managers are designed independently and

with standardized interfaces (i.e. function prototypes), NPSNET designed for the IRIX OS can be run on a PC after removing any SGI dependent manager (such as the display manager, which might consist of Performer graphics code) and replacing it with another designed to work on PCs (such as a display manager written using OpenGVS). Method abstraction ensures that manager objects can be substituted with little or no extra effort as the main application is moved between platforms.

The NPSNET application itself then becomes nothing more than a main function which starts the functional managers, identifies the intercommunications conduit by registering manager callbacks, and then loops until the program is exited, controlling (and possibly synchronizing) subcomponent operations.

An example message flow between individual managers in the networked application is illustrated in Figure 13. In this depiction, incoming network traffic (1), received by the network manager, indicating the presence of a new entity in the simulation, would be processed and sent (2) to the entity manager. The entity manager would then store a permanent record of the new entity and pass (3) information about the new entity's type and location to the display manager, which would use a platform-specific graphics engine to render the scene as it would exist with the added presence of the new entity.

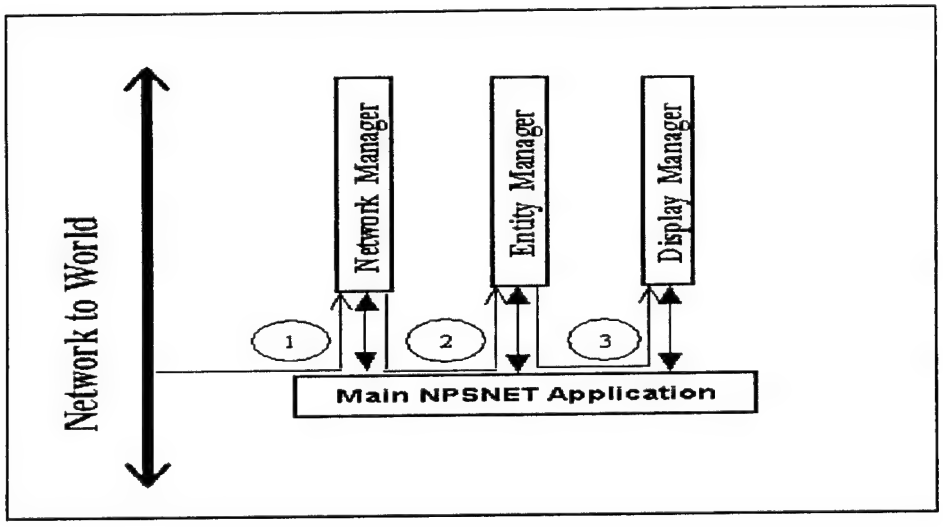


Figure 13: Sample Open NPSNET Communications Flow [BARK96]

C. DISPLAY MANAGER DESIGN

It can be easily envisioned from the example above how Open NPSNET would be ported between graphics hardware. Simply by creating display managers specifically designed for each intended destination platform, Open NPSNET would become a multi-platform application.

As an alternative to the PTG libraries, the use of display manager objects appears to present several advantages for porting NPSNET to the PC platform. First, using the OpenGVS API directly for the graphics programming, as it was designed to be used, rather than as the basis for Performer style prototypes, simplifies development and resource allocation issues. Second, OpenGVS is powerful and easy to learn. This characteristic, which made the API ideal for writing the PTG libraries, speeds implementation and makes up for having to create identically functioning software for multiple platforms. OpenGVS's

object oriented design lends itself well to this use. Third, if the requirement exists to create other platform-specific manager components for NPSNET (e.g. network managers may require hardware specific functionality) then the Open NPSNET architecture will greatly facilitate that implementation.

Appendix D contains the code for a prototype of the proposed PC Open NPSNET architecture with a minimally functional display manager. The main application is called Virtual Reality Network (VRNET). As can be seen in **`vrnet.cpp`**, the main function merely instantiates the subcomponent managers and enters a simulation loop, in which managers update themselves. The simulation loop is exited if a manager is killed. In the case of a display manager, closing the window will always cause the simulation loop to be exited.

For the prototype VRNET, only the display manager has been implemented. To validate its functionality without implementing the remaining system managers, keyboard inputs have been used to simulate the entity manager messages, but the work is intended only to demonstrate the design and provide a proof of the concept.

D. CHAPTER SUMMARY

The Open NPSNET architecture shows great promise as a course to pursue for the multi-platform version of NPSNET. Implementation abstraction will make NPSNET inherently expandable, and the direct use of platform-specific APIs for the subcomponent managers will simplify the effort.

VI. CONCLUSIONS AND FUTURE WORK

A. PROJECT CONTINUATION

As discussed in Chapter IV, the effort to bring high quality interactive graphics simulations to PCs should continue due to the clear need for the product and the promising initial performance results. However, libraries like PTG, which are capable of porting workstation specific graphics software to PCs, are most likely not the best route there due to the additional requirements of libraries for porting other platform-dependent code, such as networking functions, system functions, and the like.

A much more versatile and promising direction for the project seems to be the new Open NPSNET architecture. Its object-oriented project subcomponents are capable of separating the graphics code from other platform specific code, solving the main disadvantage of the PTG system.

As described in Chapter V, work left to be done with Open NPSNET includes the completion of the design phase, in which the inter-manager message passing conduit and standardized function prototypes will be decided upon. This phase may be particularly time consuming, as research is done to consider and evaluate the fastest, most extensible, and simplest designs to implement.

Once the design work is complete, the Open NPSNET software to drive the individual managers will still be left to be written, and in the cases where platform specifics require separate managers for each platform, considerable depth in knowledge of varying environmental specifics will be required. For example, writing display managers in

OpenGVS and Performer will require training in both those APIs, something we had initially hoped could be avoided using the cross-platform PTG libraries.

B. MULTI-PLATFORM TOOLS

Another direction for future work in the open architecture may be an NPSNET design which uses *only* multi-platform coding tools. Presently available choices for display manager building APIs include SGI's OpenGL, which has been licensed to most platforms and is rapidly becoming a de facto standard, and even Gemini Technology's OpenGVS, which was used in the PTG work for this thesis, and which proved to be an adequate tool to quickly create high quality graphics applications.

For the other managers, free multi-platform versions of networking and development software are available from Sun™ Microsystems Laboratories, the GNU Free Software Foundation under its general public license, and others. The Tool Control Language/ Toolkit (Tcl/Tk) for example may work well for the input manager, while Mesa, (an OpenGL clone) may work well for the display manager.

C. OTHER WAYS TO ACHIEVE THE MULTI-PLATFORM GOAL

Research continues in other widely varying routes to platform independence for networked virtual reality in general, although none were pursued during the course of this work. Some of these showing promise are named below.

Particularly in the field of networking, where real-time communication between entities is the goal for believable virtual environments, multicast networks, open format message protocol data units (PDUs) and the Real-time Transport Protocol (RTP) are

current areas of ongoing research. Additionally, message passing via Common Gateway Interface (cgi) scripts using HyperText Transfer Protocol (http) queries as inputs may be possible, and the crossplatform Java® language is gaining in popularity. Crossplatform Virtual Reality Modeling Language (VRML) 3D scene generation in browsers is another possible direction of future work.[BRUT95]

D. FUTURE ADVANCES IN PERSONAL COMPUTER GRAPHICS

As was predicted at the start of this project, the performance of PC hardware continues to increase while the price of the hardware continues decreased dramatically. During the course of this project alone, prices of replacement components for the NPS graphics lab PCs have dropped 40-75%, while the performance of these replacements outpace their predecessors by 15-66% in speed, and additionally provide a substantial increase in functionality [ELSA96][SCHE95]. PCs will continue become more attractive for implementing computationally expensive graphic applications, once thought of as being limited to the realm of workstations. Today, high-end PCs have entered into the low-end graphic workstation market, and are poised to redirect development in the graphic application market.[USEL96]

Large corporations, which were once solely high-end workstation driven, are now finding it cost effective to use PC based graphic workstations. The shear number of PCs, both in the home and workplace, currently generate sufficient income for hardware and software manufactures to invest heavily into research and development, further enabling them to outpace the workstation manufactures in the production of new products. The era

of workstation innovation and PC emulation is being replaced by original PC innovation funded by volume sales.[USEL96]

The focus of NPSNET should continue to move toward the PC platform. By re-designing the NPSNET architecture for multi-platform use, three-dimensional, real-time graphics will be available to a much larger audience at a fraction of the cost.

APPENDIX A
PERFORMER TO OPENGVS PORTING GUIDE

A. PTG INSTALLATION

This guide contains the steps required to properly execute the PTG OpenGVS and Performer demos. The completed project software, **`PTG.tar.gz`**, contains the source code of the PTG project, the original Performer demos and the OpenGVS examples.

The inclusion of the Performer demos serves two purposes. First, they illustrate the textual changes to the code required to use Performer programs with the OpenGVS-based PTG libraries. Second, these programs demonstrate the nearness in appearance of these new executables to the original Performer demos, demonstrating the performance of the PTG/OpenGVS libraries.

The appropriate commercially available software packages for the Performer and OpenGVS APIs must be properly installed prior to compiling and running the respective example demos. To properly install the PTG software unzip/untar the file **`PTG.tar.gz`** with the following commands:

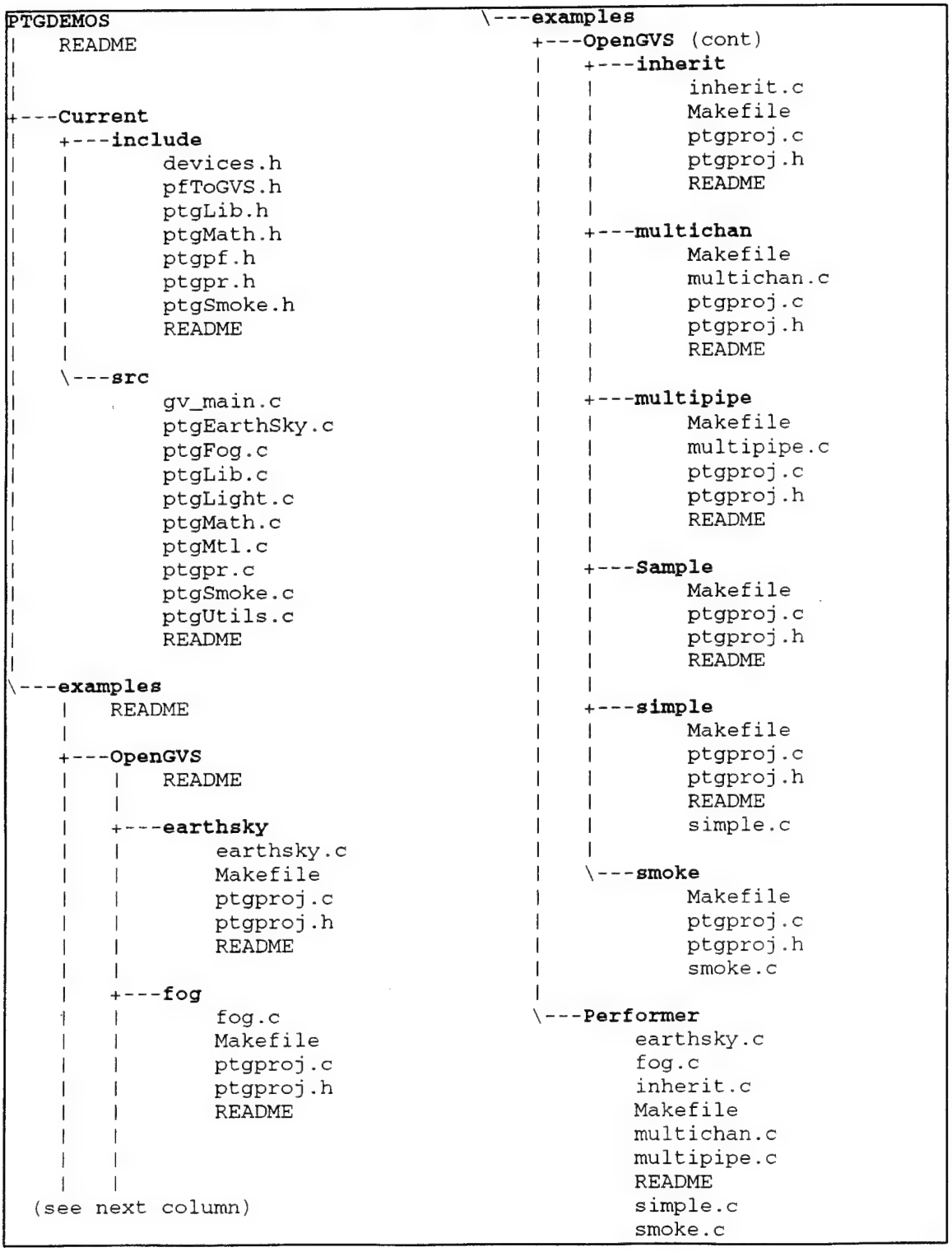
```
gunzip PTG.tar.gz

tar xvf PTG.ta
```

or if your GNU software is current

```
tar zxvf PTG.tar.gz
```

or the equivalent commands. The installation is now complete, with the resulting directory structure as shown in Figure 14.

Appendix Figure 14: PTG Directory Structure



B. USING THE PERFORMER DEMO PROGRAMS

The Performer demos may be compiled using SGI Performer 1.2 libraries. To compile the Performer demos, enter the Performer directory (`PTG/examples/Performer`) and type **`make`** to create the series of Performer executables. Consult the Performer 1.2 Users's Guide [PERF94] for more information regarding these Performer programs.

C. USING THE OPENGVS DEMO PROGRAMS

To run the existing OpenGVS demos, each of the projects will first need to be compiled separately in each demo's subdirectory due to their identical (and therefore conflicting) project file names. These demos were last built with OpenGVS V4.0-b15.

D. PORTING YOUR OWN PERFORMER PROGRAMS TO OPENGVS

To convert a Performer 1.2 program into a GVS project using the PTG porting code, several items must be modified in the Performer main program. Keep in mind that not all Performer functions have been re-written for use in OpenGVS. The emphasis of this project was NPSNET graphics code, and therefore gaps exist in the ported Performer functions.

The following modifications to the Performer main program must be made to work in OpenGVS:

1. Comment-out, delete, or include conditionally on the basis of a preprocessor directive **#include** statements that the main program contains.

2. Add the line to the top of the Performer main program file:

   #include ../Current/include/pfToGVS.h

3. Change the function **main()** to **pfMain()**.

4. If necessary, edit **pfMain()** for hardcoded paths, etc. (e.g. **pfFilePath()**)

5. Add the following line just prior to entering the Performer simulation loop: **return G_SUCCESS;**

6. Copy the simulation loop into **ptgproj.c**. Do not include the looping construct (usually "**while**").

7. Comment-out, delete, or include conditionally on the basis of a preprocessor directive the simulation loop in **pfMain()**.

8. Comment-out, delete, or include conditionally on the basis of a preprocessor directive the variable declarations for all variables with simulation loop scope.

9. Put substitute declarations for the variables that were deleted in step 8 into **ptgproj.h** and edit names in **pfMain** and in **ptgproj.h** to avoid conflicts of recurring global names.

Ensure that the correct project name and the correct paths to the source code are located in the **Makefile**. The only modifications needed for the **Makefile** to work on either an SGI or NT platform are the PTG variable values, located at the beginning of the **Makefile,** as shown below.

```
PTG_NAME = {insert name here}

PTG_CURRENT_SRC = ../../../Current/src

PTG_CUR RENT_INCL = ../../../Current/include
```

APPENDIX B
PERFORMER TO OPENGVS SOURCE CODE

This appendix contains the complete listing of the PTG source code that is compatible on both SGI and Window NT workstations. A brief description of each of the header and source files are listed below.

a. **`pfToGVS.h`** - Contains global PTG resources allocation and utility function prototypes.

b. **`ptgLib.h`** - Contains very basic Performer type definitions and basic function prototypes needed for initialization of Performer programs.

c. **`ptgMath.h`** - Contains math **`#DEFINES`**, and math type definitions from the Performer pr.h header file.

d. **`ptgSmoke.h`** - Contains Performer smoke related function prototypes.

e. **`ptgpf.h`** - Contains a portion of the Performer **`pf.h`** file that is required for the PTG project.

f. **`ptgpr.h`** - Contains a portion of the Performer **`pr.h`** file that was required for the PTG project.

g. **`gv_main.c`** - Contains the new OpenGVS main function for all PTG programs.

h. **`ptgEarthSky.c`** - Contains the Performer **`pfEarthSky`** functions that were rewritten in OpenGVS.

i. **`ptgFog.c`** - Contains the **`pfFog`** functions that were rewritten in OpenGVS.

j. **`ptgLib.c`** - Contains the basic setup functions required by most Performer programs.

k. **ptgpr.c** - Contains **pfGeoState** and material functions.

l. **ptgLight.c** - Contains the Performer **pfLight** and **pfLightSource** functions that were rewritten in OpenGVS.

m. **ptgMath.c** - Contains the necessary Performer conversion, matrix and material functions.

n. **ptgSmoke.c** - Contains the Performer **pfSmoke** functions.

o. **ptgUtils.c** - Contains OpenGVS functions necessary for the initialization and resource allocation every GVS project.

p. **devices.h** - Invoked locally, and so needed to be duplicated for Windows NT use. Contains **#DEFINE** statements for input devices. Due to proprietary source code from SGI, a source code listing cannot be displayed.

A. PTGDEMO INCLUDE FILES.

1. /PTGDemos/Current/include/pfToGVS.h

```
/*************************************************************************
*
*  pfToGVS.h --
*
* Performer to GVS function wrappers
*
*************************************************************************/

#ifndef __Globalutils__
#define __Globalutils__ extern
#endif

#ifndef __PFTOGVS_H
#define __PFTOGVS_H

#include <g_gen.h>
#include <g_sys.h>
#include <string.h>
#include <stdlib.h>
#include <g_stdlib.h>
#include <gv_sys.h>
#include <gv_user.h>
#include <g_consts.h>
#include <gv.h>
#include <time.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <g_timer.h>

#include <gv_lsr.h>
#include <gvu_isg.h>
#include <gvu_smk.h>
#include <gv_fog.h>

#include "ptgLib.h"
#include "ptgpr.h"
#include "ptgpf.h"
#include "devices.h"
#include "ptgSmoke.h"

/*
 * NPS Performer to OpengGVS Project
 *       version         date
 */
```

```
#define  PTG_GL_ver       "0.5-a6"
#define  PTG_GL_date      "15JUL96"

/* resource allocation defines */
#define AVAIL_FBFS    4L
#define AVAIL_CHNS    8L
#define AVAIL_CAMS    6L
#define AVAIL_SCNS    8L
#define AVAIL_OBIS   10L
#define AVAIL_NODES  10L
#define AVAIL_LSRS    8L   /* OpenGVS limit - do not change this */

#define AVAIL_GEOSTATES       5L
#define AVAIL_MTLS            5L
#define AVAIL_PATHS           20L
#define AVAIL_FILES           20L
#define AVAIL_EARTHSKYS       5L
#define AVAIL_FOGS            5L
#define AVAIL_SMOKES          20L


/* project globals */
__Globalutils__ int     PTG_GL_argc;
__Globalutils__ char ** PTG_GL_argv;

__Globalutils__ G_Name  PTG_GL_initName;
__Globalutils__ G_Name  PTG_GL_filePath[AVAIL_PATHS];
__Globalutils__ G_Name  PTG_GL_filePathListSet;
__Globalutils__ G_Name  PTG_GL_importCmd[AVAIL_FILES];
__Globalutils__ int     PTG_GL_filePathLength[AVAIL_PATHS];
__Globalutils__ int     PTG_GL_pathNumber;
__Globalutils__ int     PTG_GL_fileNumber;

__Globalutils__ float   PTG_GL_yon;
__Globalutils__ float   PTG_GL_hither;
__Globalutils__ float   PTG_GL_aov;
__Globalutils__ float   PTG_GL_sim_time;
__Globalutils__ int     PTG_GL_currentPlatform;

__Globalutils__ int              PTG_GL_PFNFYLEVEL_flag;
__Globalutils__ pfNotifyData     PTG_GL_NotifyData;
__Globalutils__ int              PTG_GL_NotifyThreshold;
__Globalutils__ pfNotifyFuncType PTG_GL_DefaultHandlerFunc;

__Globalutils__ GV_Rgba     PTG_GL_erase_color;
__Globalutils__ G_Position  PTG_GL_campos;
__Globalutils__ G_Rotation  PTG_GL_camrot;
__Globalutils__ GV_Viewport PTG_GL_normalized_viewport;

__Globalutils__ long   PTG_GL_allocatedFbfs;
__Globalutils__ long   PTG_GL_allocatedChns;
```

```
__Globalutils__ long   PTG_GL_allocatedCams;
__Globalutils__ long   PTG_GL_allocatedScns;
__Globalutils__ long   PTG_GL_allocatedLsrs;
__Globalutils__ long   PTG_GL_allocatedObis;
__Globalutils__ long   PTG_GL_allocatedNodes;
__Globalutils__ int    PTG_GL_allocatedGeoStates;
__Globalutils__ int    PTG_GL_allocatedMtls;
__Globalutils__ int    PTG_GL_allocatedESky;
__Globalutils__ int    PTG_GL_allocatedFog;
__Globalutils__ int    PTG_GL_allocatedSmoke;

__Globalutils__ GV_Fbf      PTG_GL_fbf[ AVAIL_FBFS ];
__Globalutils__ GV_Channel  PTG_GL_chn[ AVAIL_CHNS ];
__Globalutils__ GV_Camera   PTG_GL_cam[ AVAIL_CAMS ];
__Globalutils__ GV_Scene    PTG_GL_scn[ AVAIL_SCNS ];
__Globalutils__ GV_Light    PTG_GL_lsr[ AVAIL_LSRS ];
__Globalutils__ GV_Obi      PTG_GL_obi[ AVAIL_OBIS ];
__Globalutils__ GV_Obi      PTG_GL_node[ AVAIL_NODES ];
__Globalutils__ pfGeoState  PTG_GL_geoState[ AVAIL_GEOSTATES ];
__Globalutils__ pfMaterial  PTG_GL_material[ AVAIL_MTLS ];
__Globalutils__ pfEarthSky  PTG_GL_ESky[ AVAIL_EARTHSKYS ];
__Globalutils__ pfFog       PTG_GL_Fog[ AVAIL_FOGS ];
__Globalutils__ GV_Obi      PTG_GL_Smoke[ AVAIL_SMOKES ];

__Globalutils__ GV_Obd      PTG_GL_null_obd;

__Globalutils__ int    PTG_GL_MultiProcessMode;
__Globalutils__ int    PTG_GL_MultiPipeMode;
__Globalutils__ int    PTG_GL_Phase_mode;
__Globalutils__ double PTG_GL_Frame_rate;

/* globals modes for the pfgeostates */
__Globalutils__ long PTG_PFSTATE_ALPHAFUNC;
__Globalutils__ long PTG_PFSTATE_ALPHAREF;
__Globalutils__ long PTG_PFSTATE_ANTIALIAS;
__Globalutils__ long PTG_PFSTATE_CULLFACE;
__Globalutils__ long PTG_PFSTATE_DECAL;
__Globalutils__ long PTG_PFSTATE_ENCOLORTABLE;
__Globalutils__ long PTG_PFSTATE_ENFOG;
__Globalutils__ long PTG_PFSTATE_ENHIGHLIGHTING;
__Globalutils__ long PTG_PFSTATE_ENLIGHTING;
__Globalutils__ long PTG_PFSTATE_ENTEXTURE;
__Globalutils__ long PTG_PFSTATE_ENWIREFRAME;
__Globalutils__ long PTG_PFSTATE_SHADEMODEL;
__Globalutils__ long PTG_PFSTATE_TRANSPARENCY;

/*****************************************************/

typedef
enum{ light_type, scene_type, obinstance_type, node_type,
      group_type, unknown_type } PTG_pointer_type ;
```

```
/*
 * These are the Performer project functions
 * which must be edited to get the ptgLibrary
 * to work properly.
 */
int pfMain( int argc, char *argv[] ) ;

int GV_demo_sys( int argc, char *argv[] );

/* Initialize/cleanup before/after OpenGVS project */
void PTG_pre_pfMain( void );
void PTG_post_pfMain( void );

void PTG_GlobalInit ( void );
void PTG_GlobalClose( void );

/* Internal Utils */
void drawnull( void );
GV_Obd build_the_null_def( void );
PTG_pointer_type PTG_resource_pointer_lookup( void* );

/********current project loop globals go here****/
#include "ptgproj.h"

#endif
```

2. /PTGDemos/Current/include/ptgLib.h

```
/*****************************************************
*           NPS Performer to OpenGVS Project
* ptgLib.h
*
* This file contains definitions and prototypes
* of the Performer API which are part of the
* project to port Performer to the multi-platform
* OpenGVS API. The Performer man pages and ptgLib.c
* code should be consulted for details.
*
*****************************************************/

#ifndef __PTGLIB_H
#define __PTGLIB_H

#include "pfToGVS.h"


typedef GV_Fbf          pfPipe;
typedef GV_Channel      pfChannel;
typedef GV_Scene        pfScene;
typedef GV_Obi          pfNode;
typedef GV_Light        pfLightSource;
typedef GV_Obi          pfGroup;
typedef GV_Obi          pfDCS;

typedef GV_Fog          pfFog;
typedef GV_Light        pfLight;
typedef GV_Material     pfMaterial;


typedef float   pfVec2[2];
typedef float   pfVec3[3];
typedef float   pfVec4[4];
typedef float   pfMatrix[4][4];

extern pfMatrix pfIdentMat;

typedef struct {
    pfVec3      xyz;
    pfVec3      hpr;
} pfCoord;

typedef struct {
    pfVec3      pos;
    pfVec3      dir;
    float       length;
} pfSeg;
```

```
typedef struct {
    pfVec3      normal;
    float       offset;         /* pt dot normal = offset */
} pfPlane;

typedef struct {
    pfVec3      center;
    float       radius;
} pfSphere;

typedef struct {
    pfVec3      center;
    float       radius;
    pfVec3      axis;
    float       halfLength;
} pfCylinder;

typedef struct {
    pfVec3      min;
    pfVec3      max;
} pfBox;


/*
 * These are Performer functions rewritten to
 * invoke OpenGVS functions, "porting"
 * them to the multi-platform OpenGVS API.
 * See the Performer man pages for details.
 */
void       foreground( void );

void       prefposition( int, int, int, int );

void       winopen( char [] );

long       pfAddChild( void *, void *);

void   pfChanFOV( pfChannel *, float, float );
void   pfChanNearFar( pfChannel *, float, float );
void   pfChanScene( pfChannel *, pfScene * );
void   pfChanView( pfChannel *, pfVec3, pfVec3 );
void   pfChanViewport (pfChanne , float l, float r, float b, float t);
void   pfConfig( void );

void   pfDCSRot( pfDCS * dcs, float h, float p, float r );
void   pfDCSScale( pfDCS * dcs, float s);
void   pfDCSTrans( pfDCS * dcs, float x, float y, float z );

void   pfExit( void );
void   pfFilePath( char * );
```

```
void   pfGetChanFOV(const pfChannel* chan, float* horiz, float* vert);

const char *    pfGetFilePath( void );
long            pfGetNodeBSphere (void* , pfSphere* );
pfPipe *        pfGetPipe( long );
void *          pfGetSharedArena( void );
float           pfGetTime( void );

void     pfInit( void );
int      pfInitClock( float );
void     pfInitGfx( pfPipe * p );
void     pfInitPipe(pfPipe * pipe, void (*initFunc)(pfPipe * pipe) );

pfChannel *     pfNewChan( pfPipe *);
pfDCS *         pfNewDCS( void );
pfGroup *       pfNewGroup( void );
pfScene *       pfNewScene( void );

void            pfuExitUtil(void);
void            pfuInitUtil(void);

#if G_SYS_WIN32
void sleep( unsigned seconds );
#endif

#endif
```

3. /PTGDemos/Current/include/ptgMath.h

```
/*************************************************************
 *
 * This file contains a subset of definitions, macros and
 * prototypes from the SGI Performer header file which are needed
 * for the project to port Performer to the mulit-platform
 * OpenGVS API. The Performer man pages and prmath.h
 * source code should be consulted for details.
 * These definitions and macros were not redefined.
 *************************************************************/

/*
 * pr.h  Include file for Performer performance rendering library.
 *
 * $Revision: 1.54 $
 * $Date:  1994/03/01 00:46:40 $
 */

#ifndef __PTGMATH_H
#define __PTGMATH_H

#include "pfToGVS.h"

#define PF_X     0
#define PF_Y     1
#define PF_Z     2
#define PF_W     3
#define PF_T     3 /* Translation row in matrices */

#define PF_H     0 /* Heading */
#define PF_P     1 /* Pitch */
#define PF_R     2 /* Roll */

#define PFFP_UNIT_ROUNDOFF 1    /* unit round off */
#define PFFP_ZERO_THRESH   2    /* smaller than this is zero */
#define PFFP_TRAP_FPES     3    /* trap floating point exceptions */

#ifdef __STDC__    /* ANSI C knows about float constants */
#define PF_PI     3.14159265358979323846f /* F for SP float */
#define PF_HUGEVAL  3.40282347e+37f /* F doesn't work for 4.0.1???? */

#define PF_DEG2RAD(x) ((x)*PF_PI/180.0f)
#define PF_RAD2DEG(x) ((x)*180.0f/PF_PI)

 * Speed oriented macros
 */

/* macro for fast square roots */
/* thresholds chosen so it's no worse than pfSqrt() */
```

```
#define PF_SQRT1(_x) \
 (((_x) > 0.9996f && (_x) < 1.001f) ? \
   0.5f + 0.5f*(_x) : \
   pfSqrt(_x))

#else /* __STDC__ */
   /* CCKR C doesn't understand float constants -> slow fp */

#define PF_PI     3.14159265358979323846
#define PF_HUGEVAL  3.40282347e+37
#define PF_DEG2RAD(x) ((x)*PF_PI/180.0)
#define PF_RAD2DEG(x) ((x)*180.0/PF_PI)

/* macro for fast square roots of things probably near one */
/* thresholds chosen so it's no worse than pfSqrt() */
#define PF_SQRT1(_x) \
 (((_x) > 0.9996 && (_x) < 1.001) ? \
  0.5 + 0.5 * (_x) : \
  pfSqrt(_x))

#endif /* __STDC__ */

#define PF_SQUARE(_x) ((_x)*(_x))
#define PF_MIN2(_x1,_x2) (((_x1) < (_x2)) ? (_x1) : (_x2))
#define PF_MAX2(_x1,_x2) (((_x1) > (_x2)) ? (_x1) : (_x2))

/*------------------------------ Sin/Cosine -------------------------*/

void   pfSinCos(float arg, float* s, float* c);
float  pfTan(float arg);
float  pfArcTan2(float y, float x);
float  pfArcSin(float arg);
float  pfArcCos(float arg);
float  pfSqrt(float arg);

/*---------------------- Frame Control ----------------------------*/

void pfuTravCalcBBox(pfNode *node, pfBox *box);

/* ------------------- Vector ----------------------------------*/

void pfAddVec3(pfVec3 dst, const pfVec3 v1, const pfVec3 v2);
void pfCopyVec3(pfVec3 dst, const pfVec3 v);
void pfScaleVec3(pfVec3 dst, float s, const pfVec3 v);
void pfSetVec3( pfVec3 dst, float x, float y, float z );
void pfSubVec3(pfVec3 dst, const pfVec3 v1, const pfVec3 v2);

/* ------------------------ Matrix 4x4 ---------------------------*/
void pfMultMat(pfMatrix dst, const pfMatrix m1, const pfMatrix m2);

#endif
```

4. /PTGDemos/Current/include/ptgpf.h

```
/*************************************************************
 * ptgpf.h - This file contains a subset of the Performer pf.h file.
 * This file contains definitions and prototypes
 * of the Performer API which are part of the
 * project to port Performer to the mulit-platform
 * OpenGVS API. The Performer man pages and the ptgpf.c
 * code should be consulted for details.
 *************************************************************/
/*
  * pf.h  Include file for Performer rapid prototyping library.
 *
 * $Revision: 1.278 $
 * $Date: 1994/03/16 03:45:51 $
 *
 */

#ifndef __PTGPF_H
#define __PTGPF_H
/*------------------------- Initialization ----------------------*/

typedef void (*pfStageFuncType)(int _pipe, int _stage);

/* pfMultiprocess() */
#define PFMP_FORK_ISECT          1  /* 0x1 */
#define PFMP_FORK_CULL           2  /* 0x2 */
#define PFMP_FORK_DRAW           4  /* 0x4 */
#define PFMP_FORK_DBASE          8  /* 0x8 */
#define PFMP_CULLoDRAW          10 /* 0x10000 */  /* 65536 */
#define PFMP_CULL_DL_DRAW       20 /* 0x20000 */  /* 131072 */

#define PFMP_DEFAULT            -1
#define PFMP_APPCULLDRAW        0
#define PFMP_APPCULL_DL_DRAW    (PFMP_CULL_DL_DRAW)
#define PFMP_APPCULL_DRAW       (PFMP_FORK_DRAW)
#define PFMP_APP_CULLDRAW       (PFMP_FORK_CULL)
#define PFMP_APP_CULL_DL_DRAW   (PFMP_FORK_CULL | PFMP_CULL_DL_DRAW)
/* 131074 */
#define PFMP_APP_CULL_DRAW      (PFMP_FORK_CULL | PFMP_FORK_DRAW)
/* 6 */
#define PFMP_APPCULLoDRAW       (PFMP_FORK_DRAW | PFMP_CULLoDRAW)
/* 65540 */
#define PFMP_APP_CULLoDRAW      (PFMP_FORK_CULL | PFMP_FORK_DRAW |
                                 PFMP_CULLoDRAW) /* 65542 */

int     pfGetMultipipe(void);
int     pfMultipipe(int num);
int     pfGetMultiprocess(void);
int     pfMultiprocess(int _mpMode);
```

```
/*------------------------- Frame Control ------------------------*/

typedef void (*pfIsectFuncType)(void *_data);
typedef void (*pfDBaseFuncType)(void *_data);
typedef void (*pfSyncFuncType)(void);

/* pfPhase() */
#define PFPHASE_FLOAT         0
#define PFPHASE_LOCK          1
#define PFPHASE_FREE_RUN      2
#define PFPHASE_LIMIT         3

extern int pfSync(void);
extern int pfFrame(void);
extern void pfPhase(int _phase);
extern float pfFrameRate(float _rate);
extern float pfGetFrameRate(void);

#endif
```

5. /PTGDemos/Current/include/ptgpr.h

```
/*************************************************************
 * ptgpr.h -  This file contains a subset of the Performer pr.h file.
 * This file contains definitions and prototypes
 * of the Performer API which are part of the
 * project to port Performer to the mulit-platform
 * OpenGVS API. The Performer man pages and the ptgpr.c
 * code should be consulted for details.
 *************************************************************/

/*
 * pr.h  Include file for Performer performance rendering library.
 *
 * $Revision: 1.313 $
 * $Date:  1994/03/16 03:55:25 $
 */


#ifndef __PTGPR_H
#define __PTGPR_H

#include "pfToGVS.h"
#include "ptgMath.h"

#ifndef FALSE
#define FALSE   0
#endif

#ifndef TRUE
#define TRUE    1
#endif

#define PF_OFF  0
#define PF_ON   1

#define PF_MAXSTRING 256

 /* pfOverride() Modes, pfGStateMode(), pfGStateInherit() */
#define PFSTATE_ENLIGHTING        1/* 0x2*/
#define PFSTATE_ENTEXTURE         2/* 0x4*/
#define PFSTATE_TRANSPARENCY      3/* 0x10*/
#define PFSTATE_ALPHAFUNC         4/* 0x20*/
#define PFSTATE_ENFOG             5/* 0x40*/
#define PFSTATE_ANTIALIAS         6/* 0x100*/
#define PFSTATE_CULLFACE          7/* 0x200*/
#define PFSTATE_ENCOLORTABLE      8/* 0x400*/
#define PFSTATE_DECAL             9/* 0x4000*/
#define PFSTATE_SHADEMODEL       10/* 0x8000*/
#define PFSTATE_ENWIREFRAME      11/* 0x10000*/
```

```
#define PFSTATE_ALPHAREF        12/* 0x40000*/
#define PFSTATE_ENHIGHLIGHTING  13/* 0x100000*/

/* pfOverride() Attributes, pfGStateAttr(), pfGStateInherit() */
#define PFSTATE_FRONTMTL        14/* 0x1*/
#define PFSTATE_TEXTURE         15/* 0x8*/
#define PFSTATE_TEXENV          16/* 0x80*/
#define PFSTATE_COLORTABLE      17/* 0x800*/
#define PFSTATE_BACKMTL         18/* 0x1000*/
#define PFSTATE_FOG             19/* 0x2000*/
#define PFSTATE_LIGHTMODEL      20/* 0x20000*/
#define PFSTATE_LIGHTS          21/* 0x80000*/
#define PFSTATE_HIGHLIGHT       22/* 0x200000*/

/* pfEnable-pfDisable these PFSTATE modes */
#define PFEN_LIGHTING        2
#define PFEN_TEXTURE         3
#define PFEN_FOG             4
#define PFEN_WIREFRAME       5
#define PFEN_COLORTABLE      6
#define PFEN_HIGHLIGHTING    7


/* pfTransparency() */
#define PFTR_OFF               0
#define PFTR_ON                1
#define PFTR_FAST              2
#define PFTR_HIGH_QUALITY      3
#define PFTR_BLEND_ALPHA       4
#define PFTR_MS_ALPHA          5
#define PFTR_NO_OCCLUDE        6/* 0x100*/


/* pfAntialias() */
#define PFAA_OFF        0
#define PFAA_ON         1


#define MAX(a,b) (((a)>(b)) ? (a) : (b))
#define MIN(a,b) (((a)<(b)) ? (a) : (b))

/* OpenGVS does not distinguish between these. */
typedef pfLight pfLightModel;


/* basic structure of a material
 * colors are [4] for
 * ambient, diffuse, emmision (emmisive),  &
 * specular.
 */
/*
```

```
typedef struct
{   float    alpha;
    float    shininess;
    long     side;
    pfVec3   emission;
    pfVec3   ambient;
    pfVec3   specular;
    pfVec3   diffuse;
} _pfMaterial;
typedef _pfMaterial pfMaterial;
*/

/* structure for a geostate.  a geostate
 * contains all modes settings and a pointer
 * for each attribute of a pfState
 */
typedef struct
{
   long lighting_enable;
   long texturing_enable;
   long transparency;
   long alpha_function;
   long fogging_enable;
   long antialiasing;
   long face_culling;
   long colortable_enable;
   long decal;
   long wireframe_enable;
   long alpha_reference;
   long highlighting_enable;

   pfMaterial frontmtl;
   pfMaterial backmtl;

   pfLight *lights;
   pfLightModel *lightmodel;
} _geostate;

typedef _geostate pfGeoState;


pfNode *  LoadFile (char *, pfGeoState * );
pfNode* LoadFlt ( char* file_name );

/*------------------------- pfMaterial --------------------------*/


/* MATERIAL properties  from /usr/include/gl/glconst.h  */
#define DEFMATERIAL   0
#define EMISSION      1
#define AMBIENT       2
```

```
#define DIFFUSE         3
#define SPECULAR        4
#define SHININESS       5
#define COLORINDEXES    6
#define ALPHA           7
/*++++++++++++++++++  end of glconst.h ++++++++++++++++++++*/

#define PFMTL_AMBIENT       AMBIENT
#define PFMTL_DIFFUSE       DIFFUSE
#define PFMTL_EMISSION      EMISSION
#define PFMTL_SPECULAR      SPECULAR

#define PFMTL_FRONT           0
#define PFMTL_BACK            1
#define PFMTL_BOTH            2

#define PFMTL_CMODE_COLOR       0
#define PFMTL_CMODE_EMISSION    1
#define PFMTL_CMODE_AMBIENT     2
#define PFMTL_CMODE_DIFFUSE     3
#define PFMTL_CMODE_SPECULAR    4
#define PFMTL_CMODE_AD          5
#define PFMTL_CMODE_NULL        6

void    pfApplyMtl( pfMaterial* _mat );
float   pfGetMtlAlpha( pfMaterial* _mat );
void   pfGetMtlColor(pfMaterial* _mat, long _acolor,float* _r, float*
                     _g, float* _b);
float   pfGetMtlShininess(pfMaterial* _mat);
long    pfGetMtlSide(pfMaterial* _mat);
void    pfMtlAlpha( pfMaterial* _mat, float _alpha );
void   pfMtlColor( pfMaterial* _mat, long _acolor, float _r, float _g,
                   float _b );
void    pfMtlShininess(pfMaterial* _mat, float _shininess);
void    pfMtlSide(pfMaterial* _mat, long _side);

pfMaterial*   pfNewMtl( void * _arena );


/*----------------------- pfGeoState ----------------------------*/
void         pfGStateAttr(pfGeoState* _gs, long _attr, void* _a);
long         pfGetGStateMode(pfGeoState* _gs, long _attr);
void         pfGStateMode(pfGeoState* _gs, long _attr, long _a);
pfGeoState*  pfNewGState(void* _arena);
void         ptg_applyGeoState(pfGeoState *_geostate);

void PTG_apply_geostate(pfGeoState *geostate);

/*----------------------------- pfState ----------------------*/

/* pfOverride() Modes, pfGStateMode(), pfGStateInherit() */
```

```
#define PFSTATE_ENLIGHTING        1/* 0x2*/
#define PFSTATE_ENTEXTURE         2/* 0x4*/
#define PFSTATE_TRANSPARENCY      3/* 0x10*/
#define PFSTATE_ALPHAFUNC         4/* 0x20*/
#define PFSTATE_ENFOG             5/* 0x40*/
#define PFSTATE_ANTIALIAS         6/* 0x100*/
#define PFSTATE_CULLFACE          7/* 0x200*/
#define PFSTATE_ENCOLORTABLE      8/* 0x400*/
#define PFSTATE_DECAL             9/* 0x4000*/
#define PFSTATE_SHADEMODEL       10/* 0x8000*/
#define PFSTATE_ENWIREFRAME      11/* 0x10000*/
#define PFSTATE_ALPHAREF         12/* 0x40000*/
#define PFSTATE_ENHIGHLIGHTING   13/* 0x100000*/


/* pfOverride() Attributes, pfGStateAttr(), pfGStateInherit() */
#define PFSTATE_FRONTMTL         14/* 0x1*/
#define PFSTATE_TEXTURE          15/* 0x8*/
#define PFSTATE_TEXENV           16/* 0x80*/
#define PFSTATE_COLORTABLE       17/* 0x800*/
#define PFSTATE_BACKMTL          18/* 0x1000*/
#define PFSTATE_FOG              19/* 0x2000*/
#define PFSTATE_LIGHTMODEL       20/* 0x20000*/
#define PFSTATE_LIGHTS           21/* 0x80000*/
#define PFSTATE_HIGHLIGHT        22/* 0x200000*/


void  pfOverride(long mask, long val);

/* ------------------------------ pfAntiallias ----------------*/
void  pfAntialias(long _type);
long  pfGetAntialias(void);

/* ------------------------------ pfTransparency ---------------*/
void  pfTransparency(long _type);
long  pfGetTransparency(void);

/* ------------------------------  pfDecal() -------------------*/

#define PFDECAL_OFF                 0
#define PFDECAL_LAYER               2
#define PFDECAL_BASE                3
#define PFDECAL_BASE_FAST           4
#define PFDECAL_BASE_HIGH_QUALITY   5
#define PFDECAL_BASE_STENCIL        6
#define PFDECAL_BASE_DISPLACE       7

void pfDecal(long _mode);
long pfGetDecal(void);

/* ------------------------------pfCullFace() -------------------*/
```

```
#define PFCF_OFF            0
#define PFCF_BACK           1
#define PFCF_FRONT          2
#define PFCF_BOTH           4

void pfCullFace(long _cull);
long pfGetCullFace(void);


/* ------------------------------ pfAlphaFunc --------------------*/

/* pfAlphaFunc() */
#define PFAF_OFF            AF_ALWAYS
#define PFAF_NEVER          AF_NEVER
#define PFAF_LESS           AF_LESS
#define PFAF_EQUAL          AF_EQUAL
#define PFAF_LEQUAL         AF_LEQUAL
#define PFAF_GREATER        AF_GREATER
#define PFAF_NOTEQUAL       AF_NOTEQUAL
#define PFAF_GEQUAL         AF_GEQUAL
#define PFAF_ALWAYS         AF_ALWAYS

void pfAlphaFunc(long _ref, long _func);
void pfGetAlphaFunc(long* _ref, long* _func);

/* ---------------------------- pfLight -------------------------*/

long       pfIsLightOn( pfLight* _lt );
pfLight*   pfNewLight( void* _arena );

void   pfLightPos( pfLight* _lt, float _x, float _y, float _z,
                   float _w );
void   pfGetLightPos( pfLight* _lt, float* _x, float* _y, float* _z,
                      float* _w );
void    pfLightAmbient( pfLight* _lt, float _r, float _g, float _b );
void    pfGetLightAmbient( pfLight* _lt, float* _r, float* _g,
                           float* _b );
void    pfLightColor( pfLight* _lt, float _r, float _g, float _b );
void    pfGetSpotLightDir( pfLight* _lt, float* _x, float* _y,
                           float* _z);
void    pfSpotLightDir( pfLight* _lt, float _x, float _y, float _z );
void    pfGetLightColor( pfLight* _lt, float* _r, float* _g,
                         float* _b );
void    pfLightOff( pfLight* _lt );
void    pfLightOn( pfLight* _lt);

pfLightSource * pfNewLSource( void );

/* --------------------------- pfLightModel ---------------------*/
extern pfLightModel* pfNewLModel(void* _arena);
```

```
extern void  pfApplyLModel(pfLightModel* _lm);

/*------------------------------------------------------------------*/

void pfEnable    ( long mode );
void pfDisable   ( long mode );
long pfGetEnable ( long mode );
void ptg_applyGeoState( pfGeoState *_geostate );

/*---------------------------- pfEarthSky ----------------------*/

typedef struct
{
    /* contains four arrays for: mode, color, attribute & fog */
    long mode[2];
    float color[8][4];
    float attrib[9];

    pfChannel * assoc_channel;

} _pfEarthSky;
typedef _pfEarthSky pfEarthSky;


/* pfESkyMode() */
#define PFES_BUFFER_CLEAR 300
#define PFES_TAG          301
#define PFES_FAST         302
#define PFES_SKY          303
#define PFES_SKY_GRND     304
#define PFES_CLOUDS       305
#define PFES_OVERCAST     306
#define PFES_SKY_CLEAR    307

/* pfESkyAttr() */
#define PFES_CLOUD_TOP     310
#define PFES_CLOUD_BOT     311
#define PFES_TZONE_TOP     312
#define PFES_TZONE_BOT     313
#define PFES_GRND_FOG_TOP  316
#define PFES_HORIZ_ANGLE   317
#define PFES_GRND_HT       318

/* pfESkyColor()  */
#define PFES_SKY_TOP       350
#define PFES_SKY_BOT       351
#define PFES_HORIZ         352
#define PFES_GRND_FAR      353
#define PFES_GRND_NEAR     354
#define PFES_CLEAR         357
```

```
/* pfESkyFog() */
#define PFES_GRND          380
#define PFES_GENERAL       381

pfEarthSky* pfNewESky(void);
void pfESkyMode(pfEarthSky* _esky, long _mode, long _val);
void pfESkyAttr(pfEarthSky* _esky, long _attr, float _val);
void pfESkyColor(pfEarthSky* _esky, long _which, float _r, float _g,
                 float _b, float _a);
void pfESkyFog(pfEarthSky* _esky, long _which, pfFog* _fog);
void pfChanESky(pfChannel * chan, pfEarthSky *sky);

void ptg_apply_mode (pfEarthSky* esky);
 void ptg_apply_attrib (pfEarthSky* esky );

/* Notification stuff is from /usr/include/Performer/pr.h */
/*---------------------------- Notification ---------------------*/

/* pfNotify() severity */
#define PFNFY_ALWAYS            0
#define PFNFY_FATAL             1
#define PFNFY_WARN              2
#define PFNFY_NOTICE            3
#define PFNFY_INFO              4
#define PFNFY_DEBUG             5
#define PFNFY_FP_DEBUG          6
#define PFNFY_INTERNAL_DEBUG    7

/* pfNotify()  error */
#define PFNFY_USAGE             1
#define PFNFY_RESOURCE          2
#define PFNFY_SYSERR            3
#define PFNFY_ASSERT            4
#define PFNFY_PRINT             5
#define PFNFY_INTERNAL          6
#define PFNFY_FP_OVERFLOW       7
#define PFNFY_FP_DIVZERO        8
#define PFNFY_FP_INVALID        9
#define PFNFY_FP_UNDERFLOW     10
#define PFNFY_MORE             -1 /* Continuation of the previous err */

typedef struct
{
    int         severity;
    int         pferrno;
    char        *emsg;
} pfNotifyData;

typedef void (*pfNotifyFuncType)(pfNotifyData *);

extern void pfNotifyHandler(pfNotifyFuncType _handler);
```

```
    extern pfNotifyFuncType pfGetNotifyHandler(void);
    extern void pfDefaultNotifyHandler(pfNotifyData *notice);
    extern void pfNotifyLevel(int _severity);
    extern int pfGetNotifyLevel(void);
    extern void pfNotify(int _severity, int _error, char *_format, ...);


    /*------------------------- pfMalloc ----------------------------*/

    void* pfMalloc(size_t nbytes, void* arena);
    void* pfCalloc(size_t numelem, size_t elsize, void* arena);
    void* pfRealloc(void* ptr, size_t nbytes);
    void pfFree(void* ptr);


    /*---------------------- pf chan draw ---------------------------*/

    typedef void (*pfChanFuncType)(pfChannel* _chan, void* _userData);
    void pfChanDrawFunc(pfChannel* chan, void (*initFunc)(pfChannel *
func, void * ));

    void pfClearChan(pfChannel*);
    void pfDraw(void);
    /*------------------------ pfFog --------------------------------*/

    /* defines for fogvertex  from /usr/include/gl/glconst.h*/
    #define FG_OFF       0
    #define FG_ON        1
    #define FG_DEFINE    2
    #define FG_VTX_EXP   2 /* aka FG_DEFINE*/
    #define FG_VTX_LIN   3
    #define FG_PIX_EXP   4
    #define FG_PIX_LIN   5
    #define FG_VTX_EXP2  6
    #define FG_PIX_EXP2  7

    #define PFFOG_ON           FG_ON
    #define PFFOG_OFF          FG_OFF
    #define PFFOG_VTX_EXP      FG_VTX_EXP
    #define PFFOG_VTX_LIN      FG_VTX_LIN
    #define PFFOG_PIX_EXP      FG_PIX_EXP
    #define PFFOG_PIX_LIN      FG_PIX_LIN
    #define PFFOG_VTX_EXP2     FG_VTX_EXP2
    #define PFFOG_PIX_EXP2     FG_PIX_EXP2
    #define PFFOG_PIX_SPLINE   1000 /* FG_PIX_SPLINE */
    #define PFFOG_MAXPOINTS    14

    extern pfFog* pfNewFog(void* _arena);
    extern void   pfFogType(pfFog* _fog, long _type);
    extern long   pfGetFogType(pfFog* _fog);
    extern void   pfFogRange(pfFog* _fog, float _onset, float _opaque);
```

```
extern void   pfGetFogRange( pfFog* _fog, float* _onset,
                             float* _opaque );
extern void  pfFogOffsets( pfFog* _fog, float _onset, float _opaque );
extern void   pfGetFogOffsets( pfFog* _fog, float *_onset,
                               float * _opaque);
extern void  pfFogRamp( pfFog* _fog, long _points, float* _range,
                        float* _density, float _bias );
extern void  pfGetFogRamp( pfFog* _fog, long* _points, float* _range,
                           float* _density, float* _bias );
extern void  pfFogColor( pfFog* _fog, float _r, float _g, float _b );
extern void pfGetFogColor( pfFog* _fog, float* _r, float* _g,
                           float* _b);
extern void pfApplyFog( pfFog* _fog );
void PTG_apply_fog( void );

#endif
```

6. /PTGDemos/Current/include/ptgSmoke.h

```
/*****************************************************************
*
*  ptgSmoke.h --
*
* NPS Performer to GVS Project function wrapper library file
*  that contains the header for the conversion of pfuSmoke functions.
*
*****************************************************************/
#ifndef __PTGSMOKE_H
#define __PTGSMOKE_H
#include "pfToGVS.h"

typedef GV_Obi   pfuSmoke;

/* these are performer defines */
#define PFUSMOKE_BILLOW       0
#define PFUSMOKE_EXPLOSION    1
#define PFUSMOKE_FIRE         2 /* yes */
#define PFUSMOKE_SMOKE        3 /* yes */
#define PFUSMOKE_EXHAUST      4
#define PFUSMOKE_DUST         5 /* yes */
#define PFUSMOKE_MISSLE       6 /* yes */
#define PFUSMOKE_SMOKE_FIRE   7

#define PFUSMOKE_NUM_TYPES    8

#define PFUSMOKE_STOP         0
#define PFUSMOKE_START        1

pfuSmoke* pfuNewSmoke(void);
void pfuDrawSmokes(pfVec3 eye); /* Draw Process only */
void pfuGetSmokeDensity(pfuSmoke* smoke, float *dens, float *diss,
                        float *expansion);
void pfuGetSmokeVelocity(pfuSmoke* smoke, float *turbulence,
                         float *speed);
void pfuInitSmokes(void);
void pfuSmokeColor(pfuSmoke* smoke, pfVec3 bgn, pfVec3 end);
void pfuSmokeDensity(pfuSmoke* smoke, float dens, float diss,
                     float expansion);
void pfuSmokeDir(pfuSmoke* smoke, pfVec3 dir);
void pfuSmokeDuration(pfuSmoke* smoke, float dur);
void pfuSmokeMode(pfuSmoke* smoke, long mode);
void pfuSmokeOrigin(pfuSmoke* smoke, pfVec3 origin, float radius);
void pfuSmokeType(pfuSmoke *smoke, long type);
void pfuSmokeVelocity(pfuSmoke* smoke, float turbulence,float speed);
void PTG_apply_smoke(void );

#endif
```

B. PTGDEMO PROGRAM FILES

1. /Current/src/gv_main.c

```
/*****************************************************************
*
*  gv_main.c --
*
* Main program for PTG demos.
*
*****************************************************************/
#define __Globalutils__
#include "../include/pfToGVS.h"

/*****************************************************************
*
*  main --
*
* Establish callbacks to appropriate project specific routines
* and initialize GVS
*
*****************************************************************/
#if G_SYS_WIN32
   #if !defined GV_MAIN_WINMAIN
     #define GV_MAIN_WINMAIN 1
     #endif
   #else
     #define GV_MAIN_WINMAIN 0
#endif


#if !GV_MAIN_WINMAIN
int main( int argc, char *argv[] )
#else /* if !GV_MAIN_WINMAIN */
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow )
#endif /* if GV_MAIN_WINMAIN */
{
    int rva;

#if GV_MAIN_WINMAIN
    int argc ;
    char ** argv ;

    GVW_sys_set_main_parameters( hInstance, hPrevInstance, lpCmdLine,
                                 nCmdShow ) ;

    rva = G_prs_command_line( lpCmdLine, &argc, &argv ) ;
    if (rva == G_FAILURE)
```

```
 fprintf( stderr, "WinMain: unable to parse command line\n" ) ;
#endif /* if GV_MAIN_WINMAIN */

 PTG_GlobalInit();
 /* call to ptgUtils.c function */
    rva = GV_demo_sys( argc, argv );

    if (rva == G_FAILURE)
 return (EXIT_FAILURE);

    return (EXIT_SUCCESS);
}
```

2. /Current/src/ptgEarthSky.c

```
/*****************************************************************
*
*  ptgEarthSky.c --
*
* NPS Performer to GVS Project function wrapper library
*
*   An array of pfEarthSky members was created in pftogvs.h
*   When pfNewESky is called, the next pfEarthSky in the array
*   will be initialized to the performer values.  Each of the
*   other function calls that modify the pfEarthSky node will
*   actually be modifing the pfEarthSky in the array.  Not
*   until pfChanESky is called will the association of a
*   pfEarthSky be made with a channel.
*
*
*****************************************************************/

#include "../include/pfToGVS.h"

/* pfNewESky
 *  -- initialize a new pfearthsky structure from the
 *  array of structures.
 *  In pfToGVS.h an array of structures was created that
 *  will simulate the performer earthsky model.  The structure
 *  contains several arrays and a pointer to a pfchannel.
 *  mode[2]
 *  color[8][4]
 *  attrib[9]
 *
 *  j  23 feb 96
 */

pfEarthSky*  pfNewESky(void)
{
   /* Ensures a limited number of earthsky models are created */
   if ( PTG_GL_allocatedESky >= AVAIL_EARTHSKYS )
   {
      printf( "PTG: Fail pfNewESky. Not enough AVAIL_EARTHSKYS.\n" );
      printf( "PTG: Usable pfEarthSky 0 - %d. Exiting PTG.\n",
         AVAIL_EARTHSKYS - 1 );
      exit( G_FAILURE );
   }
   printf("PTG: Created new ESky # %i\n", PTG_GL_allocatedESky );

   return &PTG_GL_ESky[PTG_GL_allocatedESky++];
}
```

```
/*  pfEskyMode
 *   -- if the mode is valid, assigns the val to
 *  the applicable array location.
 *  mode[0] holds the values for pfes_buffer_clear
 *  mode[1] holds the values for pfes_clouds
 */
void  pfESkyMode(pfEarthSky* esky, long mode, long val)
{
   /* a mode can only be one of these two values */
   if(mode != PFES_BUFFER_CLEAR && mode != PFES_CLOUDS)
   {
      printf("PTG: pfESkyMode Failure \n");
      printf("Second parameter not in range %i\n", mode);
      exit( G_FAILURE );
   }
 /* valid vals are :PFES_FAST, TAG, SKY, SKY_GRND, SKY_CLEAR &
  * PFES_ OVERCAST
  */
   if ( val <= PFES_BUFFER_CLEAR || val > PFES_SKY_CLEAR  || val ==
PFES_CLOUDS )
   {
      printf("PTG: pfESkyMode Failure \n");
      printf("Third parameter not in range %i\n", val);
      exit( G_FAILURE );
   }

   /* store the values
      pfes_buffer_clear values will be stored in the mode [0]
      pres_overcast value will be stored in mode[1]
   */
   if( mode == PFES_BUFFER_CLEAR )
   {
      if ( val != PFES_CLOUDS )
      {
         esky->mode[ 0 ] = val;
      }
      else
      {
         printf("PTG: pfESkyMode Failure \n");
         printf("Third parameter not in range %i\n", val);
         exit( G_FAILURE );
      }
   }
   else  /* only supported mode */
      esky->mode[ 1 ] = PFES_OVERCAST;

   /* Determine if a channel has already been assigned.  If so, apply
      modes,  else apply during pfChanSky */
   if (esky->assoc_channel != NULL)
      ptg_apply_mode(esky);
}
```

```
/*  pfEskyAttr
 *   -- Stores the attribute values (height, cloud level, etc.)
 *  if the attribute is valid, function assigns the val to
 *  to the applicable array location.
 *  Attributes range from 310-318.   pfes_cloud_top = 310, used as a
 *  base to calculate the offset.  offset = attrib - base.
 *  See the performer man pages for valid attributes.
 *  j
 */
void  pfESkyAttr(pfEarthSky* esky, long attrib, float val)
{
   if ( ( attrib < PFES_CLOUD_TOP ) || ( attrib > PFES_GRND_HT ) ||
      ( ( attrib < PFES_GRND_FOG_TOP) && ( attrib > PFES_TZONE_BOT ) ) )
   {
      printf("PTG: pfESkyMode Failure \n");
      printf("Second parameter not in range\n");
      exit( G_FAILURE );
   }

/* use the base (pfes_cloud_top) to calculate the offset of the array */
   esky->attrib[attrib - PFES_CLOUD_TOP ] = val;


   /* Determine if a channel has already been assigned.  If so, apply
      modes,  else apply during pfChanSky */
    if (esky->assoc_channel != NULL)
   {
      ptg_apply_mode( esky );
      ptg_apply_attrib( esky );
   }
}


/*  pfEskycolor
 *  -- if valid which_color (cloud top, bottom, etc.), assigns the color
 *  the applicable color storage location.
 *  stores the colors for:
 *  pfes_sky_top, bot, horiz, grnd_far, near, cloud_bot, top, clear
 *  j t 28 feb96
 */
void  pfESkyColor(pfEarthSky* esky, long which_color,
                            float r, float g, float b, float a)
{
   int ix = 0;

   if ( ( ( which_color < PFES_SKY_TOP ) || ( which_color > PFES_CLEAR )
)
      && ( which_color != PFES_CLOUD_BOT) &&
         ( which_color != PFES_CLOUD_TOP ) )
```

```
    {
        printf("PTG: pfESkyMode Failure \n");
        printf("Second parameter not in range\n");
        exit( G_FAILURE );
    }

    /* ensures that max is 1.0 */
    if ( r > 1.0 ) r = 1.0;
    if ( b > 1.0 ) b = 1.0;
    if ( g > 1.0 ) g = 1.0;
    if ( a > 1.0 ) a = 1.0;

    ix = which_color - PFES_SKY_TOP;

    /* PFES_CLOUD_BOT & TOP are defined as 310 & 311
     * so slight modification is needed.
     * PFES_SKY_TOP defined as 350, PFES_CLEAR defined as 357
     */
    if ( which_color == PFES_CLOUD_BOT )
        ix = 6;
    else if ( which_color == PFES_CLOUD_TOP )
        ix = 5;

    esky->color[ix][0] = r;
    esky->color[ix][1] = g;
    esky->color[ix][2] = b;
    esky->color[ix][3] = a;


    /* Determine if a channel has already been assigned.  If so, apply
       modes,  else apply during pfChanSky */

     if (esky->assoc_channel != NULL)
    {
        ptg_apply_mode( esky );
        ptg_apply_attrib( esky );
    }
}



/* pfESkyFog
 *  used when pfes_grnd_fog_top is > 0. (default )
 * once things are up, treat this as another fog with base of 0.
 */
void pfESkyFog( pfEarthSky* esky, long which, pfFog* fog )
{
    ( esky, fog );
    switch ( which )
    {
        case ( PFES_GRND ):
```

```
      { break;
      }

      case ( PFES_GENERAL ):
      { break;
      }
      default: break;
   }
}


/*  pfChanESky
 *  assigns the sky to the channel
 *  and creates the earth sky model in gvs
 *  j  28 feb 96
 */
void pfChanESky( pfChannel * chan, pfEarthSky *sky )
{
   int EarthSkyNumber = 0;
   int found = FALSE;

   /* ensure that earthsky has been created */
   while ( found == FALSE && EarthSkyNumber < AVAIL_EARTHSKYS )
   {
      if ( sky == &PTG_GL_ESky[ EarthSkyNumber ] )
         found = TRUE;
      else
       EarthSkyNumber++;
   }

   /* cannot display earth/sky without initial states */
   if (found == FALSE)
   {
      printf("pfEarthSky not found");
      exit( G_FAILURE );
   }

   /* stores the associated channel to allow for dynamic modification */
   sky->assoc_channel  = chan;

   /* set up infinite ground/sky model in gvs */
   GVU_isg_setup( (*(*sky).assoc_channel) );

   /* apply the modes, attributes and colors for the earthsky model */
   ptg_apply_attrib( sky );
   ptg_apply_mode( sky );


}
```

```
/* ptg_apply_mode
 * apply the specified mode & colors to the earthsky model
 * is called every time the mode changes after a
 * channel has been created. j - 19mar96
 */
void ptg_apply_mode (pfEarthSky* sky )
{
   int number_of_points = 1;           /* performer only uses 1 point */
   G_State status;
   float angle[3] = {10.0};
   GV_Rgba  diffuse, sky_color[3];

   GVU_isg_inq_state( (*(*sky).assoc_channel), &status);

   /* turn off the earthsky state for the channel if it is on */
   if (status == G_ON)
      GVU_isg_set_state ( (*(*sky).assoc_channel), G_OFF );

   /* turn sky/ground on to reset channel */
   GVU_isg_set_state ( (*(*sky).assoc_channel), G_ON );

   /* turn the sky/ground state off to modify them */
   GVU_isg_set_land_state( (*(*sky).assoc_channel), G_OFF );
   GVU_isg_set_sky_state( (*(*sky).assoc_channel), G_OFF );
   GVU_isg_set_land_horizon_state( (*(*sky).assoc_channel), G_OFF );

   switch ( sky->mode[ 0 ] )     /* PFES_BUFFER_CLEAR */
   {
      case ( PFES_FAST ):
      { /* This mode will use the pfes_clear color for background */

         sky_color[0].r = sky->color[PFES_CLEAR - PFES_SKY_TOP][0];
         sky_color[0].g = sky->color[PFES_CLEAR - PFES_SKY_TOP][1];
         sky_color[0].b = sky->color[PFES_CLEAR - PFES_SKY_TOP][2];
         sky_color[0].a = sky->color[PFES_CLEAR - PFES_SKY_TOP][3];

         /* use the clear color as sky will emulate the pfes_fast mode
            of performer
          */
        GVU_isg_set_sky( (*(*sky).assoc_channel), 1, angle, sky_color );
        GVU_isg_set_sky_state( (*(*sky).assoc_channel), G_ON );
        GV_chn_set_erase_mode( (*(*sky).assoc_channel),
           GV_CHN_ERASE_MODE_FAST );

         break;
      }
      case( PFES_TAG ):
      {
        /* reserved for reality engine */
             break;
      }
```

```
case( PFES_SKY ):    /* only sky and horizon is displayed */
{

   /* sky color always use pfes_sky_top  */
   sky_color[0].r = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][0];
   sky_color[0].g = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][1];
   sky_color[0].b = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][2];
   sky_color[0].a = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][3];

   /* set sky color  */
   GVU_isg_set_sky( (*(*sky).assoc_channel), 1, angle,
     sky_color );

  /* turn sky on */
  GVU_isg_set_sky_state( (*(*sky).assoc_channel) , G_ON );

  /* turn horizon on */
  GVU_isg_set_land_horizon_state ( (*(*sky).assoc_channel),
                                   G_ON );
  GV_chn_set_erase_mode( (*(*sky).assoc_channel),
                         GV_CHN_ERASE_MODE_ON );
   break;
}

case( PFES_SKY_GRND ):  /* both sky and ground to be displayed */
{
    /* sky color always use pfes_sky_top  */

   sky_color[0].r = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][0];
   sky_color[0].g = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][1];
   sky_color[0].b = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][2];
   sky_color[0].a = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][3];

   /* land color is always pfes_grnd_far */
   diffuse.r = sky->color[PFES_GRND_FAR - PFES_SKY_TOP][0];
   diffuse.g = sky->color[PFES_GRND_FAR - PFES_SKY_TOP][1];
   diffuse.b = sky->color[PFES_GRND_FAR - PFES_SKY_TOP][2];
   diffuse.a = sky->color[PFES_GRND_FAR - PFES_SKY_TOP][3];

  /* set land color */
  GVU_isg_set_land_color( (*(*sky).assoc_channel), &diffuse,
                          &diffuse );

  /* turn horizon on */
  GVU_isg_set_land_horizon_state ( (*(*sky).assoc_channel),
                                   G_ON );

  /* enable land to be seen */
  GVU_isg_set_land_state( (*(*sky).assoc_channel), G_ON );
```

```
        /* enable sky to be seen */
        GVU_isg_set_sky_state( (*(*sky).assoc_channel), G_ON );
        GV_chn_set_erase_mode( (*(*sky).assoc_channel),
                               GV_CHN_ERASE_MODE_ON );

        break;
      }

     /* only sky is rendered */
      case ( PFES_SKY_CLEAR ):
      {
         /* sky color always use pfes_sky_top  */
         sky_color[0].r = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][0];
         sky_color[0].g = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][1];
         sky_color[0].b = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][2];
         sky_color[0].a = sky->color[PFES_SKY_TOP - PFES_SKY_TOP][3];

         /* set sky color  */
         GVU_isg_set_sky( (*(*sky).assoc_channel) , number_of_points,
                          angle, sky_color );

         /* turn sky state on */
         GVU_isg_set_sky_state( (*(*sky).assoc_channel), G_ON );
         GV_chn_set_erase_mode( (*(*sky).assoc_channel),
                                GV_CHN_ERASE_MODE_ON );

         break;
      }
   }
}


/* ptg_apply_attrib
 * apply the specified mode to the earthsky model
 * after a channel has been created.
 * performer only has one cloud available, gvs supports multiple cloud
"decks"
 * in PTG, only deck[0] will be used.
 */
void ptg_apply_attrib ( pfEarthSky* sky )
{
   float upper_cloud_height  = sky->attrib[ PFES_CLOUD_TOP -
                               PFES_CLOUD_TOP ],
         lower_cloud_height  = sky->attrib[ PFES_CLOUD_BOT -
                               PFES_CLOUD_TOP ],
         upper_scud_extent   = sky->attrib[ PFES_TZONE_TOP -
                               PFES_CLOUD_TOP ],
         lower_scud_extent   = sky->attrib[ PFES_TZONE_BOT -
                               PFES_CLOUD_TOP ];

   GV_Rgba top_color, bottom_color;
```

```
    /* using 305 as base to get the offset off cloud_bot & top */
    /* color range is from 350 - 357 & pfes_cloud_top is 310 */
    /* could have easily indexed the array with 5 & 6 */

    top_color.r =  sky->color[ PFES_CLOUD_TOP - 305 ][0];
    top_color.g =  sky->color[ PFES_CLOUD_TOP - 305 ][1];
    top_color.b =  sky->color[ PFES_CLOUD_TOP - 305 ][2];
    top_color.a =  sky->color[ PFES_CLOUD_TOP - 305 ][3];

    bottom_color.r =  sky->color[ PFES_CLOUD_BOT - 305 ][0];
    bottom_color.g =  sky->color[ PFES_CLOUD_BOT - 305 ][1];
    bottom_color.b =  sky->color[ PFES_CLOUD_BOT - 305 ][2];
    bottom_color.a =  sky->color[ PFES_CLOUD_BOT - 305 ][3];

    /*  this section sets the cloud states.  in performer only one
     *  cloud can be usedat one time.  in gvs the first
     *   cloud is the #0 deck.  in the following
     *  formulas 0 is being used to represent the 0 cloud deck.
     */

    /* only display a cloud if the top is higher than the bottom */
    if ( upper_cloud_height > lower_cloud_height )
    {

        GVU_isg_set_deck_extent( (*(*sky).assoc_channel), 0,
           lower_cloud_height, upper_cloud_height  );
        GVU_isg_set_deck_color( (*(*sky).assoc_channel), 0,
           &bottom_color, &top_color );
        GVU_isg_set_deck_state( (*(*sky).assoc_channel), 0, G_ON );
    }
      else
          GVU_isg_set_deck_state( (*(*sky).assoc_channel),0, G_OFF );

    /*  tzone_top & bottom correspond to the transition layer above and
     * below the cloud deck.  If the value of the transition layer
     *  pfes_tzone_top is < cloud_top, then the transition is disabled.
     */

    GVU_isg_set_deck_scud_extent( (*(*sky).assoc_channel), 0,
          lower_scud_extent, upper_scud_extent);
}
```

3. /Current/src/ptgFog.c

```
/*****************************************************************
*
*  ptgFog.c --
*
* NPS Performer to GVS Project function wrapper library
*  File contains the functions required to simulate the perfomer
*  pfFog functions.
*
*****************************************************************/

#include "../include/pfToGVS.h"


/* pfNewFog
 * creates a new fog and sets up initial
 * default type.  Performer default
 * j
 */
pfFog* pfNewFog(void* _arena)
{
    _arena;
   if ( PTG_GL_allocatedFog >= AVAIL_FOGS )
   {
     printf( "PTG: Fail pfFog. Not enough AVAIL_FOGS.\n" );
     printf( "PTG: Usable pfFog 0 - %d. Exiting PTG.\n",
             AVAIL_FOGS - 1 );
      exit( G_FAILURE );
   }
   GV_fog_create( &PTG_GL_Fog[ PTG_GL_allocatedFog ] );
   GV_fog_set_type( PTG_GL_Fog[ PTG_GL_allocatedFog],
                    GV_FOG_TYPE_EXP2  );

   printf( "PTG: FOG Model %d created.\n", PTG_GL_allocatedFog );
   return ( &PTG_GL_Fog[ PTG_GL_allocatedFog++] );
}


/* pfFogType
 * -- sets the type of the fog, density, etc.
 *
 */
void pfFogType(pfFog* fog, long type)
{
   int FogNumber = 0;
   int found = FALSE;

  /* ensure that fog has been created */
   while ( found == FALSE && FogNumber <= AVAIL_FOGS )
```

```
{
   if ( fog == &PTG_GL_Fog[ FogNumber ] )
      found = TRUE;
   else
      FogNumber++;
}

/* cannot display earth/sky without initial states */
if (found == FALSE)
{
   printf("pfFog not found");
   exit( G_FAILURE );
}

/* set the fog state */
switch ( type )
{
   case (PFFOG_VTX_LIN ):
   {
      GV_fog_set_type( *fog, GV_FOG_TYPE_LINEAR  );
      break;
   }
   case (PFFOG_VTX_EXP ) :
   {
      GV_fog_set_type( *fog, GV_FOG_TYPE_EXP  );
      break;
   }
   case (PFFOG_VTX_EXP2 ) : /* default */
   {
      GV_fog_set_type( *fog, GV_FOG_TYPE_EXP2 );
      break;
   }
   case (PFFOG_PIX_LIN ) :
   {
      printf("PTG: FOG using GV_FOG_TYPE_LINEAR\n");
      GV_fog_set_type( *fog, GV_FOG_TYPE_LINEAR  );
      break;
   }
    case (PFFOG_PIX_EXP ) :
   {
      printf("PTG: FOG using GV_FOG_TYPE_EXP\n");
      GV_fog_set_type( *fog,GV_FOG_TYPE_EXP  );
      break;
   }
   case (PFFOG_PIX_EXP2 ) :
   {
      printf("PTG: FOG using GV_FOG_TYPE_EXP2\n");
      GV_fog_set_type( *fog,  GV_FOG_TYPE_EXP2 );
      break;
   }
```

```
      case (PFFOG_PIX_SPLINE ) :
      /* GVS does not support this type of fog */
      {  /* will use the following */
         printf("PTG Warning: Using GV_FOG_TYPE_EXP2\n");
         GV_fog_set_type( *fog,  GV_FOG_TYPE_EXP2 );
         break;
      }
   }
}


/*  pfGetFogType
 *  returns the fog type
 */
long  pfGetFogType(pfFog* fog)
{
   int type;

   GV_fog_inq_type( *fog, &type );
   return (long) type;
}


/* pfFogRange
 * Sets the fog range (start & farthest distance )
 */
void  pfFogRange(pfFog* fog, float onset, float opaque)
{
   int which;

   GV_fog_inq_type( *fog, &which );

   GV_fog_set_start_distance( *fog, onset );
   GV_fog_set_opaque_distance ( *fog, opaque );
}


/* pfGetFogRange -- returns the onset (start distance)
 * and opaque (point where completely blended)
 *
 */
void  pfGetFogRange(pfFog* fog, float* onset, float* opaque)
{
   GV_fog_set_start_distance( *fog, *onset );
   GV_fog_set_opaque_distance ( *fog, *opaque );
}


/*   pfFogOffsets
 *   -- used with PFFOG_PIX_SPLINE
 * no-op
```

```
 */
void  pfFogOffsets(pfFog* _fog, float _onset, float _opaque)
{
   ( _fog,  _onset,  _opaque);
}


/*   pfGetFogOffsets
 *   -- used with PFFOG_PIX_SPLINE
 *  no-op
 */
void  pfGetFogOffsets(pfFog* _fog, float *_onset, float *_opaque)
{
      ( _fog,  _onset,  _opaque);
}


/*  pfFogRamp
 *  Sets the ramp of the fog model
 *  set of points to modify the fog
 *
 */
void  pfFogRamp(pfFog* fog, long points, float* range,
      float* density, float bias)
{
   bias;
   /* bias is always ignored in performer */
   GV_fog_set_opacity_ramp( *fog, points, range, density );
}


/*  pfFogColor
 *  sets the fog color
 *
 *  j t
 */
void pfFogColor( pfFog* fog, float r, float g, float b )
{
   GV_Rgba color;

   color.r = r;
   color.g = g;
   color.b = b;
   color.a = 0.0;

    GV_fog_set_color ( *fog, &color );
}
```

```
/*  pfGetFogColor
 *  returns the fog color
 *
 */
 void pfGetFogColor( pfFog* fog, float* r, float* g, float* b)
{
   GV_Rgba color;

   GV_fog_inq_color( *fog, &color );

   *r =  color.r;
   *g =  color.g;
   *b =  color.b;
}


/* pfApplyFog -- performer and GVS use fog similar,
 * except that performer allows the programmer to
 * apply a fog without defining a channel.  Because
 * of this, PTG_GL_Fog[0] was choosen to be the current
 * active fog prior to defining a channel.  if this
 * function is called with a active channel, fog is
 * applied, else fog is applied in pfPostMain to the
 * active channels.
 *
 */
void   pfApplyFog( pfFog* fog )
{
   GV_Channel channel;

   GV_chn_inq_current ( &channel );

   if ( channel != NULL )
   {
      GV_fog_set_state( *fog, G_ON );
      GV_chn_set_fog( channel, *fog );
   }
   else
   {
       printf("PTG: Channel not defined \n");
       PTG_GL_Fog [ 0 ] =  *fog;
    }
}


/*  PTG_apply_fog
 *  associates all the fog model to all channels that are available
 *  prior to starting the simulation loop
 *  j t 3/12/96
 */
```

```
void PTG_apply_fog( void )
{
   int ix = 0;

   if (PTG_GL_Fog [ 0 ] != NULL )
   {
      GV_fog_set_state( PTG_GL_Fog [ 0 ], G_ON);
      for (ix =0; ix <  PTG_GL_allocatedChns; ix++)
      {
         GV_chn_set_fog( PTG_GL_chn[ ix ], PTG_GL_Fog [ 0 ] );
      }
    }
}
```

4. /Current/src/ptgLib.c

```
/*****************************************************************************
*
*  ptgLib.c --
*
* NPS Performer to GVS Project function wrapper library
*
*
*
*****************************************************************************/

#include "../include/pfToGVS.h"


/* foreground - from man page:
 *    prevents a graphical process from being put into the
 *    background.
 * f - 25jan96
 */
void foreground( void )
{
}


/*
 *  Loadfile  -  concatenates file path with a filename
 *     for single file import for use with GV_cmd.
 *     Objects are then imported for use in the sim.
 *     if the _file contains the path, then function will not
 *     search the other paths for file, and only attempt to
 *     load the file with the path given on the cmd line
 *  j - 3feb96
 */
pfNode *LoadFile (char *_file, pfGeoState *_geostate)
{
   GV_Obd           objDef;         /* A GVS object definition*/

   int ix, cmdLineLength, pathNumber = -1;
   int FileLoaded = G_FAILURE;
   char * temp;
   static int objectNumber =0;
   char buffer[5];
   G_Name objectName ="object";

 if(_geostate != NULL )
     PTG_apply_geostate(_geostate);


   printf("PTG: LoadFile -> get object %s\n", _file);
```

```
  while ( ( ++pathNumber <= PTG_GL_pathNumber ) &&
          ( FileLoaded != G_SUCCESS ) &&
          ( PTG_GL_fileNumber <= AVAIL_FILES ) )
  {
     cmdLineLength = 0;

     /* ensure that cmdline is blank */
     for( ix = 0; ix < G_NAME_LENGTH; ix++ )
        PTG_GL_importCmd[ PTG_GL_fileNumber ][ ix ]= 0;

     cmdLineLength = sprintf( PTG_GL_importCmd[ PTG_GL_fileNumber ],
                     "import file=" );
     if ((temp = strpbrk( _file, "\\/:") ) == NULL )
     {
        /* add path to command */
        cmdLineLength += sprintf(( PTG_GL_importCmd[PTG_GL_fileNumber ]
                         + cmdLineLength ),
           PTG_GL_filePath[ pathNumber ] );

        /* add end of path to command */
        if ( PTG_GL_importCmd[ PTG_GL_fileNumber][cmdLineLength-1 ]
             != '/' )
           PTG_GL_importCmd[ PTG_GL_fileNumber ][ cmdLineLength++ ]
              = '/';
     }
     else
/* file name contains the path also, do not attempt multiple paths */
        pathNumber = PTG_GL_pathNumber;

     /* add file name to command */
     cmdLineLength += sprintf( PTG_GL_importCmd[ PTG_GL_fileNumber ]
                      + cmdLineLength, _file );

     /* search previously loaded files for requested file first */
     ix = 0;
     while ( ( ix < PTG_GL_fileNumber ) && ( FileLoaded != 0 ) )
     {
        if( strncmp( PTG_GL_importCmd[ ix++ ], PTG_GL_importCmd
            [ PTG_GL_fileNumber ], cmdLineLength )==0)
           FileLoaded = G_SUCCESS;
     }

     if( FileLoaded == G_SUCCESS )
     {
        temp = strpbrk( PTG_GL_importCmd[ ix-1 ], "=" );
        temp = strpbrk( ++temp, "=" );

        FileLoaded = GV_obd_inq_by_name( ++temp, &objDef ) ||
          GV_obi_instance( objDef,
             &PTG_GL_node[ PTG_GL_allocatedNodes] );
```

```
      }
      else
      {
         /* file not loaded, must load file */
         /* add remainder of import command */
         cmdLineLength += sprintf( PTG_GL_importCmd[ PTG_GL_fileNumber ]
                          + cmdLineLength, " name=object" );
         for (ix = 0;ix < 5;ix++) buffer[ix] = '\0';

         /* convert unique number to string */
         sprintf( buffer, "%d",objectNumber );

         /* attatch number string to object name */
         cmdLineLength += sprintf( PTG_GL_importCmd[ PTG_GL_fileNumber ]
                          + cmdLineLength, buffer );

         sprintf( objectName+6,buffer );

         /* execute import command */
         FileLoaded = ( GV_cmd_service(
            PTG_GL_importCmd[ PTG_GL_fileNumber ] ) ||
                     GV_obd_inq_by_name(objectName, &objDef ) ||
                     GV_obi_instance( objDef,
                        &PTG_GL_node[ PTG_GL_allocatedNodes ] ) );

         if( FileLoaded != G_SUCCESS )
         {
             printf( "PTG: LoadFile -> failure \n" );
         }
         else
         {
            PTG_GL_fileNumber++;
            objectNumber++;
         }
      }
   }
    if( FileLoaded != G_SUCCESS )
         {
            printf( "PTG: Path or filename is not correct. \n" );
            exit(G_FAILURE);
         }

   return (pfNode*)&PTG_GL_node[ PTG_GL_allocatedNodes++ ];
}


/*
 * pfAddChild - Performer uses this function to put
 *     many types together, but OpenGVS adds to displays
 *     with separate functions for all possible combinations of node
types.
```

```
 *      To make this function work then, we must test for types
 *      first. While this may be time consuming, it will be
 *      infrequent, and actually rare during sim run-time.
 *      I think.
 *      Currently supported: scenes, obis, lights, nodes, groups, dcs
 *      f-16feb96
 */
long pfAddChild( void * parentVoid, void * childVoid)
{
    PTG_pointer_type parentType = unknown_type;
    PTG_pointer_type childType  = unknown_type;

    printf( "PTG: pfAddChild -> parent & child pointer lookup\n" );
    /* find the parent type  */
    parentType = PTG_resource_pointer_lookup( parentVoid );
    /* find the child type   */
    childType = PTG_resource_pointer_lookup( childVoid );

    /*
     * action here only if arg types are found
     */
    if( parentType == scene_type && childType == light_type )
    {
       GV_scn_add_light( *( (GV_Scene*)parentVoid ),
                         *( (GV_Light*)childVoid ) );
       printf( "PTG:    Add light to scene\n" );
       return G_SUCCESS;
    }
    if( parentType == scene_type && childType == obinstance_type )
    {
       GV_scn_add_object( *( (GV_Scene*)parentVoid ),
                          *( (GV_Obi*)childVoid ) );
       printf( "PTG:    Add object instance to scene\n" );
       return G_SUCCESS;
    }
    if( parentType == scene_type && childType == node_type )
    {
       GV_scn_add_object  ( *( (GV_Scene*)parentVoid ),
                            *( (GV_Obi*)childVoid ) );
       printf( "PTG:    Add node to scene\n" );
       return G_SUCCESS;
    }
    if( parentType == obinstance_type && childType == node_type )
    {
     GV_obi_copy( *( (GV_Obi*)childVoid ),( (GV_Obi*)parentVoid ) );
       printf( "PTG:    Associate/copy node to instance/group\n" );
       return G_SUCCESS;
    }
    if( parentType == obinstance_type && childType == obinstance_type )
    {
       GV_obi_attach_child( *( (GV_Obi*)parentVoid ),
```

```
                                *( (GV_Obi*)childVoid ) );
        printf( "PTG:    Attach object instance to parent instance\n" );
        return G_SUCCESS;
    }


    /*
     * enter here only if any arg type(s) unknown (i.e. did not
     * already return)
     */
    if( parentType == unknown_type )
    {
        printf( "PTG: pfAddChild fails -> parentType == unknown\n");
    }
    if( childType == unknown_type )
    {
        printf( "PTG: pfAddChild fails -> childType == unknown\n");
    }
    return G_FAILURE;

}



/*
 * pfChanFOV
 *    set aov with deg/rad conversion
 *    Follow Performer rules (i.e. aov between 0-180 deg)
 * f - t - 26Jan96
 */
void pfChanFOV( pfChannel * ch, float f1, float f2 )
{
    GV_Camera camera;
    float aov;

    aov = ( ( ( f1 <= 0.05 ) || ( f1 >= 179.0 ) ) ?
              f2 * G_DEG_TO_RAD : f1 * G_DEG_TO_RAD );
    printf( "PTG: pfChanFov -> assign aov %f.\n", aov/G_DEG_TO_RAD );
    GV_chn_inq_camera( *ch, &camera);
    GV_cam_set_aov( camera, aov);
}



/*
 * pfChanNearFar
 *    set clipping planes
 * f - t - 26Jan96
 */
void pfChanNearFar( pfChannel * ch, float f1, float f2 )
{
```

```
   PTG_GL_hither = f1;
   PTG_GL_yon = f2;

   GV_chn_set_clip_near( *ch, PTG_GL_hither );
   GV_chn_set_clip_far( *ch, PTG_GL_yon );
}


/*
 * pfChanScene -
 *    associate chan with scene
 *
 * f - t - 26Jan96
 */
void pfChanScene( pfChannel * ch, pfScene * sc )
{
   printf( "PTG: pfChanScene -> adding scene to channel.\n" );
   GV_chn_set_scene  ( *ch, *sc );
}



/*
 * pfChanView
 *    set camera paraeters for input pfChannel
 * f - t - 29Jan96
 */
void pfChanView( pfChannel * ch, pfVec3 xyz, pfVec3 hpr )
{
   G_Position position;
   G_Rotation rotation;

   GV_Camera camera;
   GV_chn_inq_camera( *ch, &camera );
   position.x = xyz[0];
   position.y = xyz[2];
   position.z = -xyz[1];

   rotation.x = hpr[1] * G_DEG_TO_RAD;
   rotation.y = hpr[0] * G_DEG_TO_RAD;
   rotation.z = hpr[2] * G_DEG_TO_RAD;

   GV_cam_set_position( camera, PTG_GL_currentPlatform, &position );
   GV_cam_set_rotation( camera, PTG_GL_currentPlatform, &rotation );
}


void pfChanViewport (pfChannel* ch, float l, float r, float b, float t)
{
   GV_Viewport normalized_viewport;
```

```
   normalized_viewport.xmin = l * 2.0 - 1.0;
   normalized_viewport.xmax = r * 2.0 - 1.0;
   normalized_viewport.ymin = b * 2.0 - 1.0;
   normalized_viewport.ymax = t * 2.0 - 1.0;

   GV_chn_set_viewport( *ch, &normalized_viewport );
}


/*
 *  pfConfig - This Performer call is not required in OpenGVS.
 *      In the current version, a single processor will handle
 *      in software a multi-fbf application.
 *  f - 2feb96
 */
void pfConfig( void )
{
}



/* pfDCSRot - rotate an Obi/DCS
 *    Convert deg/rad & coord systems.
 * f - 3feb96
 */
void pfDCSRot( pfDCS * dcs, float h, float p, float r )
{
   G_Rotation rotation;

   rotation.x = p * G_DEG_TO_RAD;
   rotation.y = h * G_DEG_TO_RAD;
   rotation.z = r * G_DEG_TO_RAD;

   GV_obi_set_rotation( *dcs, &rotation );
}



/* pfDCSScale - Performer scales in proportion. OpenGVS
 *    allows axial scaling. This implementation produces
 *    the Performer functionality.
 * f - 2feb96
 */
void pfDCSScale( pfDCS * dcs, float s)
{
   G_Scaling scaling;

   scaling.x = s;
   scaling.y = s;
   scaling.z = s;
```

```
   GV_obi_set_scaling( *dcs, &scaling );
}


/* pfDCSTrans - translate an Obi/DCS
 *    Convert coord systems.
 * f - 3feb96
 */
void pfDCSTrans( pfDCS * dcs, float x, float y, float z )
{
   G_Position position;

   position.x = x;
   position.y = z;
   position.z = -y;

   GV_obi_set_position( *dcs, &position );
}



/*
 *  pfExit - should never be reached, because it's in pfMain
 *     after the sim loop, which is exited using PTG.
 *     If it is, however, exit PTG.
 *  f - 1feb96
 */
void pfExit( void )
{
   PTG_GlobalClose();
}


/*************************************************************************
 * pfFilePath
 *    store path to input object files
 *    supports multiple file complete paths
 *    such as:
 *    pfFilePath("e:/:./:e:/gemini/gv/gvm");
 *    pfFilePath("g:");
 *    would be 4 complete paths.
 * j - 3feb96
 *************************************************************************/
void pfFilePath( char * path )
{
   int ix = 0, templength = 0, totalPathLength = strlen(path);

   G_Name parsedPath;
   char *temp = path;

   sprintf( (char*)PTG_GL_filePathListSet, path );
```

```
   do
   {
      for ( ix = 0; ix < G_NAME_LENGTH; ix++ ) parsedPath [ ix ] = '\0';

      ix = 0;

      PTG_GL_pathNumber++;

      temp = strpbrk( temp, ":");

      while ( &path [ templength ]  != &temp[0] &&
              templength < totalPathLength)
      {
         if ( (&path [ templength + 1] == &temp [ 0 ] )
              && (ix == 0) && (path[ templength ] != '.') )
            temp = strpbrk( ++temp, ":");
         parsedPath [ ix++ ] = path[ templength++ ];
      }

      PTG_GL_filePathLength[ PTG_GL_pathNumber ] =
         sprintf( PTG_GL_filePath[ PTG_GL_pathNumber ], parsedPath );


      PTG_GL_filePath[ PTG_GL_pathNumber ][ PTG_GL_filePathLength[
PTG_GL_pathNumber ] ] = 0;
   }while ( ( templength++ < totalPathLength ) & ( temp++ != NULL ) );

  }



/*
 *  pfFrame - Performer requires this call to cull and draw
 *     every time a new frame is desired. A user_proc callback
 *     in OpenGVS is called everytime a new frame is desired,
 *     eliminating the need for this.
 *  f - 2feb96
 */
extern int pfFrame( void )
{
 return G_SUCCESS;
}


/*
 *  pfFrameRate - Performer uses this call to set a max
 *    frame rate, along with pfPhase. It also is intended
 *    to return the actual rate used, but this functionality
 *    is not yet implemented in PTG.
 *  f - nt
 */
```

```
extern float pfFrameRate(float _rate)
{
   PTG_GL_Frame_rate = _rate;
   G_timer_set_frame_period_const( (double)( 1/_rate) );
   if( PTG_GL_Phase_mode == PFPHASE_LIMIT )
      G_timer_set_realtime_state( G_OFF );
   printf("PTG: pfFrameRate(float) -> return value invalid.\n");
   return _rate;
}


/* pfGetChanFOV -
 *
 *
 *
 */
void pfGetChanFOV(const pfChannel* chan, float* horiz, float* vert)
{
   GV_chn_inq_aov_actual( *chan, horiz, vert);
   *horiz = *horiz * G_RAD_TO_DEG;
   *vert = *vert * G_RAD_TO_DEG;
}



/*
 *  pfGetFilePath - Return previously set FilePath.
 * f 27mar96
 */
const char * pfGetFilePath( void )
{
    return (char*)PTG_GL_filePathListSet;
}


/*
 *  pfGetFrameRate - Return previously set FrameRate.
 *    Returns -1.0 if this has not been set. Displaying
 *    statistics onscreen shows this function "works", but
 *    GVS framerate seems not to vary when set with realtime state off.
 * f  26mar96
 */
extern float pfGetFrameRate(void)
{
   return PTG_GL_Frame_rate;
}
/*
 * pfGetNodeBSphere
 *    This function uses GV_Bbox to approximate pfSphere. There is no
 * Sphere in OpenGVS, and the floating point math used here gets us
 * within 1% of the radius Performer generated. Close enough for now.
```

```
 * f - 13mar96
 */
long pfGetNodeBSphere (void* pointer, pfSphere* sphere)
{

   GV_Obi obihdl;
   GV_Bbox bbox_out;
   GV_Bbox_status bbox_status;

   float x_dim, y_dim, z_dim, max_dim = 0;
   PTG_pointer_type pointerType = unknown_type;
   max_dim;
   printf("PTG: pfGetNodeBSphere -> pointer lookup.\n");
   pointerType = PTG_resource_pointer_lookup( pointer );

   if ( pointerType == unknown_type )
   {
      printf("PTG:    unknown pointer arg.\n");
      return G_FAILURE;
   }
   if( pointerType == scene_type )
   {
      GV_scn_inq_object_first( *((GV_Scene*)(pointer)), &obihdl );
   }

   GV_obi_inq_bbox_full_world( obihdl, &bbox_out, &bbox_status );

   sphere->center[0] = ( bbox_out.xmax - bbox_out.xmin ) / 2
                       + bbox_out.xmin;
   sphere->center[1] = -( ( bbox_out.zmax - bbox_out.zmin ) / 2
                      + bbox_out.zmin );
   sphere->center[2] = ( bbox_out.ymax - bbox_out.ymin ) / 2
                     + bbox_out.ymin;

   x_dim = bbox_out.xmax - bbox_out.xmin;
   y_dim = bbox_out.ymax - bbox_out.ymin;
   z_dim = bbox_out.zmax - bbox_out.zmin;

   sphere->radius = pfSqrt( x_dim * x_dim + y_dim * y_dim + z_dim
                            * z_dim )/2.0;

   printf("PTG:    radius = %f.\n",sphere->radius);
   printf("PTG:    ctr x = %f, y = %f, z = %f\n",
      sphere->center[0],sphere->center[1],sphere->center[2] );

   return G_SUCCESS;
}

/*
 *   pfGetMultiprocess and pfGetMultipipe return the multiprocess mode
 *   and number of pfPipes configured.
```

```
 */
int pfGetMultipipe(void)
{
   return PTG_GL_MultiPipeMode;
}
int pfGetMultiprocess(void)
{
   return PTG_GL_MultiProcessMode;
}


/* pfGetPipe -
 *    pfPipes are associated with a long.
 *    Return the pfPipe associated with the long, or
 *    fail the whole system if not avail. (Can always reset
 *    AVAIL_FBFS to a higher number & try again.) If no
 *    current pfPipe is associated with a valid long, create
 *    it and return it's address.
 *    pfPipes in PTG get a parent channel added automatically.
 *    All other channels are added to this parent. See pfNewChan.
 *    f - 1feb96
 */
pfPipe * pfGetPipe( long n )
{
   if ( n >= AVAIL_FBFS )
   {
      printf( "PTG: Fail pfGetPipe. Not enough AVAIL_FBFS.\n" );
      printf( "PTG: Usable pfPipes 0 - %d. Exiting PTG.\n",
              AVAIL_FBFS - 1 );
      exit( G_FAILURE );
   }

   if ( !PTG_GL_fbf[ n ] )
   {
      GV_fbf_create( &PTG_GL_fbf[ n ] );
      PTG_GL_allocatedFbfs++;
   }
   else
   {
      printf( "PTG: Fail pfGetPipe. pfPipe numbered %d already
               assigned.\n", n );
      exit( G_FAILURE );
   }

   if ( PTG_GL_allocatedChns >= AVAIL_CHNS )
   {
      printf( "PTG: Fail pfGetPipe. Not enough AVAIL_CHNS for
               automatic parent channel.\n" );
      printf( "PTG: Usable pfChannels 0 - %d. Exiting PTG.\n",
               AVAIL_CHNS - 1 );
      exit( G_FAILURE );
```

```
    }
    GV_chn_create( &PTG_GL_chn[ PTG_GL_allocatedChns ] );

    /* set defaults for this parent (non-draw-in) channel */
    GV_chn_set_viewport( PTG_GL_chn[ PTG_GL_allocatedChns ],
&PTG_GL_normalized_viewport );
    GV_chn_set_name( PTG_GL_chn[ PTG_GL_allocatedChns ],
PTG_GL_initName );

    printf( "PTG: pfGetPipe -> adding first channel to pipe.\n" );
    GV_fbf_add_channel( PTG_GL_fbf[ n ], PTG_GL_chn[
PTG_GL_allocatedChns++ ] );

    return &PTG_GL_fbf[ n ];
}


/* pfGetTime -
 *    return elapsed sim time, or -1.0 if fails
 * f - 25jan96
 */
float pfGetTime( void )
{
 double elapsed_time;

    if ( G_timer_inq_time( &elapsed_time ) == G_SUCCESS )
    {
       return ( (float)elapsed_time - PTG_GL_sim_time );
    }

    return -1.0;

}



/* pfGetSharedArena -
 *    no Performer functionality for shared memory
 * f - 25jan96
 */
void * pfGetSharedArena( void )
{
    void * m = NULL;
    return m;   /* supress error, only */
}


/* pfInit - Since both OpenGVS and PTG are
 *   already running by the time this is called,
 *   simply return from this call.
 * f - 25jan96
```

```
 */
void pfInit( void )
{
}


/* pfInitClock - Set the global PTG_GL_sim_time
 *    since some apps require relative sim timing.
 * f- 25jan96
 */
int pfInitClock( float startTime )
{
   if ( startTime >= 0.0 )
   {
      PTG_GL_sim_time = startTime;
      return G_SUCCESS;
   }
   return G_FAILURE;
}


void pfInitGfx( pfPipe * p )
{
   p;
}


/*
 * pfInitPipe -
 *    Setup a "window" or pfPipe using the user function
 *    initFunc. initFunc is provided by the Performer user.
 * f- 25jan96
 */
void pfInitPipe(pfPipe * pipe, void (*initFunc)(pfPipe * pipe) )
{
   if( initFunc ) initFunc( pipe );
}



/* pfMultiipe - sets global
 * not needed in OpenGVS implementation.
 * f - 25jan96
 */
int pfMultipipe ( int num )
{
   PTG_GL_MultiPipeMode = num;
   return num; /* NOT EXPECTED PER PERFORMER 2.0 */
}
```

```
/* pfMultiprocess - sets global
 * not needed in OpenGVS implementation.
 * f - 25jan96
 */
int pfMultiprocess( int _mpMode )
{
   PTG_GL_MultiProcessMode = _mpMode;
   return _mpMode; /* NOT EXPECTED PER PERFORMER 2.0 */
}


/* pfNewChan - Since Performer associates channels
 *    with their pipes from the creation of the former,
 *    bookkeeping is simplified. Cameras are non-existent
 *    in Performer, so their creation and association can all
 *    be done together. PTG does not take advantage of the
 *    platform mounting capabilities of OpenGVS since
 *    Performer cannot. Note, though, that from the outset,
 *    channels created here will not be mounted to pfPipes but to
 *    the pfPipe's first channel, the parent.
 *    This function checks for that root/parent channel for the
 *    arg fbf, and adds the channel as a sub-channel to it.
 * f- 2feb96
 */
pfChannel * pfNewChan( pfPipe * p )
{
   GV_Channel channel;
   GV_Fbf fbf;
   int ix;
   int parentFound = 0;

   if ( PTG_GL_allocatedChns >= AVAIL_CHNS )
   {
      printf( "PTG: Fail pfNewChan. Not enough AVAIL_CHNS.\n" );
      printf( "PTG: Usable pfChannels 0 - %d. Exiting PTG.\n",
AVAIL_CHNS - 1 );
      exit( G_FAILURE );
   }

   GV_chn_create( &PTG_GL_chn[ PTG_GL_allocatedChns ] );
   GV_cam_create( &PTG_GL_cam[ PTG_GL_allocatedCams ] );
   GV_chn_set_camera ( PTG_GL_chn[ PTG_GL_allocatedChns ],
      PTG_GL_cam[ PTG_GL_allocatedCams++ ] );

   /* set defaults for this draw-in channel */
   GV_cam_set_aov( PTG_GL_cam[ PTG_GL_allocatedCams - 1 ],
                   PTG_GL_aov * G_DEG_TO_RAD);
   GV_chn_set_erase_color( PTG_GL_chn[ PTG_GL_allocatedChns ],
                           &PTG_GL_erase_color );
   GV_chn_set_clip_near( PTG_GL_chn[ PTG_GL_allocatedChns ],
                         PTG_GL_hither );
```

```
    GV_chn_set_clip_far( PTG_GL_chn[ PTG_GL_allocatedChns ],
                         PTG_GL_yon );

    printf( "PTG: pfNewChan -> adding new channel to parent.\n" );
    /* find the fbf's root channel, so we can add this new sub-channel */
    for( ix = 0; ix < PTG_GL_allocatedChns; ix++ )
    {
       GV_chn_inq_fbf_parent( PTG_GL_chn[ ix ], &fbf );
       if( *p == fbf )
       {
          channel = PTG_GL_chn[ ix ];
          printf( "PTG:    pfNewChan parent channel found.\n" );
          ix = PTG_GL_allocatedChns;
          parentFound = 1;
       }
    }

    if( !parentFound )
    {
       printf( "PTG: pfNewChan fails. Parent channel not found.\n" );
       exit(1);
    }

    printf( "PTG:    pfNewChan adding sub-channel to parent
                     complete.\n" );
    GV_chn_add_channel( channel, PTG_GL_chn[ PTG_GL_allocatedChns ]);

    return &PTG_GL_chn[ PTG_GL_allocatedChns++ ];
}


/* pfNewDCS - Unlike chans, cams, scenes, etc, OpenGVS does
 *    not "new" or "create" GV_Obi's. This is only done as
 *    a by-product of an association with an Obd. So, a null
 *    is associated to ensure that the Obi can be used as in
 *    Performer, prior to the final target object def being set.
 * f- 3feb96
 */
pfDCS * pfNewDCS( void )
{
    if ( PTG_GL_allocatedObis >= AVAIL_OBIS )
    {
       printf( "PTG: Fail pfNewDCS. Not enough AVAIL_OBIS.\n" );
       printf( "PTG: Usable pfDCSs 0 - %d. Exiting PTG.\n",
                AVAIL_OBIS - 1 );
       exit( G_FAILURE );
    }
    GV_obi_instance( PTG_GL_null_obd,
                     &PTG_GL_obi[ PTG_GL_allocatedObis ] );
    return &PTG_GL_obi[ PTG_GL_allocatedObis++ ];
}
```

```
/* pfNewGroup - Unlike chans, cams, scenes, etc, OpenGVS does
 *    not "new" or "create" GV_Obi's. This is only done as
 *    a by-product of an association with an Obd. So, a null
 *    is associated to ensure that the Obi can be used as in
 *    Performer, prior to the final target object def being set.
 * f- 3feb96
 */
pfGroup * pfNewGroup( void )
{
   if ( PTG_GL_allocatedObis >= AVAIL_OBIS )
   {
      printf( "PTG: Fail pfNewGroup. Not enough AVAIL_OBIS.\n" );
      printf( "PTG: Usable pfGroups 0 - %d. Exiting PTG.\n",
               AVAIL_OBIS - 1 );
      exit( G_FAILURE );
   }

  GV_obi_instance( PTG_GL_null_obd, &PTG_GL_obi[ PTG_GL_allocatedObis ]
);
   return &PTG_GL_obi[ PTG_GL_allocatedObis++ ];
}


/* pfNewScene - Performer and OpenGVS create
 *    scenes identically.
 * f - 25jan96
 */
pfScene * pfNewScene( void )
{
   if ( PTG_GL_allocatedScns >= AVAIL_SCNS )
   {
      printf( "PTG: Fail pfNewScene. Not enough AVAIL_SCNS.\n" );
      printf( "PTG: Usable pfScenes 0 - %d. Exiting PTG.\n",
               AVAIL_SCNS - 1 );
      exit( G_FAILURE );
   }

   GV_scn_create( &PTG_GL_scn[ PTG_GL_allocatedScns ] );
   return &PTG_GL_scn[ PTG_GL_allocatedScns++ ];
}


/* pfPhase -
 *   pfPhase specifies the synchronization method
 * used by the drawing process and pfSync.
 * In PTG, LIMIT will fix framerate and turn it's mode to
 * on. f-nt
 */
extern void pfPhase(int _phase)
```

```
{
   /* choices for _phase:
    *   PFPHASE_FLOAT, PFPHASE_LOCK,
    *   PFPHASE_FREE_RUN, PFPHASE_LIMIT
    */

   /* assign a max framerate */
   if( _phase == PFPHASE_LIMIT )
   {
      PTG_GL_Phase_mode = _phase;
      if( PTG_GL_Frame_rate > 0.0 )
         G_timer_set_realtime_state( G_OFF );
   }
   else
   {
      PTG_GL_Phase_mode = _phase;
      G_timer_set_realtime_state( G_ON );
   }
}


/* pfSync -
 *   In Performer, app is caused to sleep if "ahead" to
 * ensure frame rate. Not used in PTG.
 */
extern int pfSync( void )
{
   int retVal = 1;
   return retVal;
}

/* pfuInitUtil & pfuExitUtil are not implemented for PTG.
 *    The fol is an exerpt from Performer man pages:
 *    pfuInitUtil must be called before making any calls to the utility
 *    library.  pfuInitUtil creates a pfDataPool which libpfutil uses
 *    for multiprocess operation.  The pfDataPool is created in
 *    "/usr/tmp"or the directory specified by the environment variable,
 *    PFTMPDIR, if it is set.pfuExitUtil removes utility library
 *    pfDataPool from the file system.
 */
void pfuInitUtil(void)
{
}
void pfuExitUtil(void)
{
}


/*  prefposition() - assign window size.
 *  Performer uses pixel distances from lower left.
 *  OpenGVS uses -1.0 to 1.0 as full screen (0.0 is center.)
```

```
 *  The following line will be "global" to user_init file, not just
 *  function:PTG_GL_normalized_viewport;
 *  If more than 1 window (pfPipe/GV_Fbf) exists, this function
 *  will effect only the last one created.
 *  f - 2feb
 */
void prefposition( int xFrom, int xTo, int yFrom, int yTo )
{
   int width, height;
   GV_Fbf fbf;
   GV_Channel parentChannel, lastChannel;

   GV_fbf_create( &fbf );
   GV_fbf_inq_display_size( fbf, &width, &height );

   /* ensure minimum size window, and assign */
   if ( ( 0 <= xFrom ) && ( xTo > xFrom + 100 )
     && ( 0 <= yFrom ) && ( yTo > yFrom + 100 ) )
   {
  PTG_GL_normalized_viewport.xmin = xFrom * 2.0 / width  - 1.0;
  PTG_GL_normalized_viewport.xmax =   xTo * 2.0 / width  - 1.0;
  PTG_GL_normalized_viewport.ymin = yFrom * 2.0 / height - 1.0;
  PTG_GL_normalized_viewport.ymax =   yTo * 2.0 / height - 1.0;
   }

   GV_chn_inq_last( &lastChannel );
   GV_chn_inq_root( lastChannel, &parentChannel );

   GV_chn_set_viewport( parentChannel, &PTG_GL_normalized_viewport );
   GV_chn_set_erase_color( parentChannel, &PTG_GL_erase_color );

   GV_fbf_free( fbf );
}


/* sleep - put app on hold. Mimics unix functionality.
 * f- 3feb96
 */
#if G_SYS_WIN32
void sleep( unsigned seconds )
{
   G_sleep( seconds * 1000 );
}
#endif


/*  winopen() - assign window title. This title will be assigned
 *     to the last parent channel/fbf pair in the database.
 *  f - 2feb96
 */
void winopen( char windowTitle[] )
```

```
{
   GV_Channel parentChannel, lastChannel;

   strncpy( PTG_GL_initName, windowTitle, G_NAME_LENGTH );
   PTG_GL_initName[ G_NAME_LENGTH ] = 0;

   GV_chn_inq_last( &lastChannel );
   GV_chn_inq_root( lastChannel, &parentChannel );
   GV_chn_set_name( parentChannel, PTG_GL_initName );
}

/*
 *  LoadFlt
 *  Loads .flt files by calling LoadFile.
 *
 * j t 4/19/96
 */
pfNode* LoadFlt ( char* file_name )
{
    return ( LoadFile( file_name, NULL));
}
```

5. /Current/src/ptgpr.c

```
/*************************************************************
 * ptgpr.c
 *
 * This file contains the source code
 * of the Performer API which are part of the
 * project to port Performer to the multi-platform
 * OpenGVS API. The Performer man pages should be
 * consulted for details.
 *************************************************************/

#include "../include/pfToGVS.h"


/* pfGeoState functions
 *
 * increments the number of geostates that the system has.
 * returns the address to the geostate array
 * choose to start off at -1 inorder to always have the last geostate
 * avail without having to -1.  initializes the data that is contained
 * in the geostate (modes & attributes)
 */
pfGeoState * pfNewGState( void * arena)
{
   arena;

   if ( PTG_GL_allocatedGeoStates >= AVAIL_GEOSTATES )
      return NULL;

   printf("***PTG:  current geostate is %i \n",
PTG_GL_allocatedGeoStates);

   /* --- modes inherit from the global values -- */
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].alpha_function      =
PTG_PFSTATE_ALPHAFUNC;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].alpha_reference     =
PTG_PFSTATE_ALPHAREF;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].antialiasing        =
PTG_PFSTATE_ANTIALIAS;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].colortable_enable   =
PTG_PFSTATE_ENCOLORTABLE;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].decal               =
PTG_PFSTATE_DECAL;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].face_culling        =
PTG_PFSTATE_CULLFACE;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].fogging_enable      =
PTG_PFSTATE_ENFOG;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].highlighting_enable =
PTG_PFSTATE_ENHIGHLIGHTING;
```

```
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].lighting_enable      =
PTG_PFSTATE_ENLIGHTING;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].texturing_enable     =
PTG_PFSTATE_ENTEXTURE;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].transparency         =
PTG_PFSTATE_TRANSPARENCY;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].wireframe_enable     =
PTG_PFSTATE_ENWIREFRAME;

   /* set the attributes of the geostate to NULL */
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].lights       = NULL;
   PTG_GL_geoState[PTG_GL_allocatedGeoStates].lightmodel   = NULL;



  return (pfGeoState*)&PTG_GL_geoState[ PTG_GL_allocatedGeoStates++ ];
}


/*  copy_materials
 *  copies settings from one material to another
 *  j t 02mar96
 */
void copy_materials(pfMaterial mtl, GV_Material gv_mtl)
{
   static GV_Rgba color;

   GV_mtl_inq_ambient( gv_mtl, &color );
   GV_mtl_set_ambient( mtl, &color );

   GV_mtl_inq_diffuse( gv_mtl, &color );
   GV_mtl_set_diffuse( mtl, &color );

   GV_mtl_inq_emission( gv_mtl, &color );
   GV_mtl_set_emission( mtl, &color );

   GV_mtl_inq_specular( gv_mtl, &color );
   GV_mtl_set_specular( mtl, &color );
}


/* pfGStateAttr
 * Sets the specified attribute to a value
 * currently only the material (front & back) attributes work
 * j 02 mar96
 */
void pfGStateAttr( pfGeoState * gstate, long attr, void *a )
{
   pfMaterial *mtl;

   printf("PTG: pfGstateAttr :\n" );
```

```
    switch (attr)
    {
       case (PFSTATE_TEXTURE):
       {
          break;
       }
       case (PFSTATE_TEXENV ):
       {
          break;
       }
       case (PFSTATE_COLORTABLE ):
       {
             break;
       }
       case (PFSTATE_FOG ):
       {
             break;
       }
       case (PFSTATE_LIGHTMODEL ):
       {
             break;
       }
       case (PFSTATE_LIGHTS ):
       {
             break;
       }
       case (PFSTATE_HIGHLIGHT ):
       {
             break;
       }
       case (PFSTATE_FRONTMTL):
       {
          mtl = (pfMaterial*) a;
          copy_materials(gstate->frontmtl,  *mtl);
          break;
       }
       case (PFSTATE_BACKMTL):
       {
          mtl = (pfMaterial*) a;
          copy_materials(gstate->backmtl, *mtl);
       }
    }
}


/*  pfGStateMode
 *  sets the mode of the geostate
 *  j t 02mar96
 */
void pfGStateMode( pfGeoState * gstate, long mode, long val )
```

```
{
   switch (mode)
   {
      case (PFSTATE_ENTEXTURE):
         {  gstate->texturing_enable        = val; break; }
      case (PFSTATE_TRANSPARENCY):
         {  gstate->transparency            = val; break; }
      case (PFSTATE_ALPHAFUNC):
         {  gstate->alpha_function          = val; break; }
      case (PFSTATE_ENFOG):
         {  gstate->fogging_enable          = val; break; }
      case (PFSTATE_ANTIALIAS):
         {  gstate->antialiasing            = val; break; }
      case (PFSTATE_CULLFACE):
         {  gstate->face_culling            = val; break; }
      case (PFSTATE_ENCOLORTABLE):
         {  gstate->colortable_enable       = val; break; }
      case (PFSTATE_DECAL):
         {  gstate->decal                   = val; break; }
     case (PFSTATE_ENWIREFRAME):
         {  gstate->wireframe_enable        = val; break; }
      case (PFSTATE_ALPHAREF):
         {  gstate->alpha_reference         = val; break; }
      case (PFSTATE_ENHIGHLIGHTING):
         {  gstate->highlighting_enable     = val; break; }
      default : break;
   }
}


/*   pfGetGStateMode
 *   returns the value of a particular mode
 *   j t 02 mar96
 */
long pfGetGStateMode( pfGeoState * gstate, long mode)
{
   switch (mode)
   {
      case (PFSTATE_ENTEXTURE):
         return gstate->texturing_enable;
      case (PFSTATE_TRANSPARENCY):
         return gstate->transparency;
      case (PFSTATE_ALPHAFUNC):
         return gstate->alpha_function;
      case (PFSTATE_ENFOG):
         return gstate->fogging_enable;
      case (PFSTATE_ANTIALIAS):
         return gstate->antialiasing;
      case (PFSTATE_CULLFACE):
         return gstate->face_culling;
      case (PFSTATE_ENCOLORTABLE):
```

```
            return gstate->colortable_enable;
         case (PFSTATE_DECAL):
            return gstate->decal;
         case (PFSTATE_ENWIREFRAME):
            return gstate->wireframe_enable;
         case (PFSTATE_ALPHAREF):
            return gstate->alpha_reference;
         case (PFSTATE_ENHIGHLIGHTING):
            return gstate->highlighting_enable ;
         default: return -1;
      }
}

/* pfTransparency
 * sets the global transparency value
 *
 */
void pfTransparency (long type )
{
   /* can only have these values, defined in ptgpr.h */
   if (( type >= PFTR_OFF ) &&  ( type <= PFTR_MS_ALPHA ))
   {
     PTG_PFSTATE_TRANSPARENCY = type;
   }
}

/* pfTransparency
 * returns the global transparency value
 *
 */
long pfGetTransparency ( void )
{
   return PTG_PFSTATE_TRANSPARENCY;
}


/* pfAntialias
 * sets the hardware antialising mode
 *
 */
void pfAntialias( long type )
{  /* can only be on or off */
   if ( ( type == PFAA_OFF) || ( type == PFAA_ON ) )
      PTG_PFSTATE_ANTIALIAS = type;
}

 /* pfAntialias
 * returns the global antialias value
 *
 */
long pfGetAntialias(void)
```

```
{
   return  PTG_PFSTATE_ANTIALIAS;
}


/* pfDecal
 * sets the global decal value
 *
 */
void pfDecal(long mode)
{
   /* can only have these values, defined in ptgpr.h */
   if ( ( mode >= PFDECAL_OFF ) &&  ( mode <= PFDECAL_LAYER ))
   {
     PTG_PFSTATE_DECAL = mode;
   }

}


/* pfGetDecal
 * returns the global decal value
 *
 */
long pfGetDecal(void)
{
   return PTG_PFSTATE_DECAL;
}


/* pfCullFace
 * sets the global pfCullFace value
 *
 */

void pfCullFace(long cull)
{
    /* can only have these values, defined in ptgpr.h */
   if ( ( cull >= PFCF_OFF ) &&  ( cull <= PFCF_BOTH ) )
   {
     PTG_PFSTATE_CULLFACE = cull;
   }

}

/* pfGetCullFace
 * returns the global pfGetCullFace value
 *
 */
long pfGetCullFace(void)
{
```

```
   return  PTG_PFSTATE_CULLFACE;
}



/* pfAlphaFunc
 *  no-op
 */
void pfAlphaFunc(long ref, long func)
{
   ref;
   func;
}

/* pfGetAlphaFunc
 * no-op
 */
void pfGetAlphaFunc(long* _ref, long* _func)
{
   ( _ref,  _func);
}

/*
 *  turns on hardware texturing in performer
 *  Opengvs does it in software,
 *  function only provides necessary compatability for
 *  program to operate correctly.
 *
 */
void pfEnable( long mode )
{
   switch(mode)
   {
      case ( PFEN_LIGHTING ):
         { PTG_PFSTATE_ENLIGHTING      = PF_ON; break; }
      case ( PFEN_TEXTURE ):
         { PTG_PFSTATE_ENTEXTURE       = PF_ON; break; }
      case ( PFEN_FOG ):
         { PTG_PFSTATE_ENFOG           = PF_ON; break; }
      case ( PFEN_WIREFRAME ):
         { PTG_PFSTATE_ENWIREFRAME     = PF_ON; break; }
      case ( PFEN_COLORTABLE ):
         { PTG_PFSTATE_ENCOLORTABLE    = PF_ON; break; }
      case ( PFEN_HIGHLIGHTING ):
         { PTG_PFSTATE_ENHIGHLIGHTING  = PF_ON; break; }
      default: exit( G_FAILURE ); break;
   }
}
```

```
/*
 *  turns off hardware texturing in performer
 *  Opengvs does it in software,
 *  function only provides necessary compatability for
 *  program to operate correctly.
 *
 */
void pfDisable( long mode )
{
switch( mode )
   {
      case ( PFEN_LIGHTING ):
         { PTG_PFSTATE_ENLIGHTING      = PF_OFF; break; }
      case ( PFEN_TEXTURE ):
         { PTG_PFSTATE_ENTEXTURE       = PF_OFF; break; }
      case ( PFEN_FOG ):
         { PTG_PFSTATE_ENFOG           = PF_OFF; break; }
      case ( PFEN_WIREFRAME ):
         { PTG_PFSTATE_ENWIREFRAME     = PF_OFF; break; }
      case ( PFEN_COLORTABLE ):
         { PTG_PFSTATE_ENCOLORTABLE    = PF_OFF; break; }
      case ( PFEN_HIGHLIGHTING ):
         { PTG_PFSTATE_ENHIGHLIGHTING  = PF_OFF; break; }
      default: exit( G_FAILURE ); break;
   }
}


/*
 *  returns hardware texturing modes
 *
 *  function only provides necessary compatability for
 *  program to operate correctly.
 *
 */
long pfGetEnable( long mode )
{
   switch( mode )
   {
      case ( PFEN_LIGHTING ):
         {  return PTG_PFSTATE_ENLIGHTING;    }
      case ( PFEN_TEXTURE ):
         {  return PTG_PFSTATE_ENTEXTURE;     }
      case ( PFEN_FOG ):
         {  return PTG_PFSTATE_ENFOG;         }
      case ( PFEN_WIREFRAME ) :
         {  return PTG_PFSTATE_ENWIREFRAME;   }
      case ( PFEN_COLORTABLE ):
         {  return PTG_PFSTATE_ENCOLORTABLE;  }
      case ( PFEN_HIGHLIGHTING ):
         {  return PTG_PFSTATE_ENHIGHLIGHTING;}
```

```
      default:  return FALSE;
   }
}


/*
 *  pfMalloc
 *  Performer can malloc an area from a shared space. GVS does not
 *  support this.
 */
void* pfMalloc(size_t nbytes, void* arena)
{
  if( arena != NULL )
  {
     printf( "PTG: pfMalloc -> warning. Shared memory not
             implemented.\n");
     printf( "PTG:    malloc-ing from heap.\n");
  }
  return ( G_malloc ( nbytes ) );
}

/*  pfCalloc
 *  Performer can calloc an area from a shared space. GVS does not
 *  support this.
 */
void* pfCalloc( size_t numelem, size_t elsize, void* arena )
{
  if( arena != NULL )
  {
     printf( "PTG: pfCalloc -> warning. Shared memory not
              implemented.\n" );
     printf( "PTG:    calloc-ing from heap.\n");
  }
  return ( G_calloc(  numelem, elsize ) );
}


void* pfRealloc( void* ptr, size_t nbytes )
{
     return ( G_realloc(  ptr, nbytes ) );
}



void pfFree( void* ptr )
{
   G_free( ptr );
}


/*
```

```
 * pfChanDrawFunc
 *
 */
void pfChanDrawFunc(pfChannel* chan, void (*initFunc)(pfChannel *
func, void * ) )
{
   if( initFunc ) initFunc( chan, NULL );

}

/* pfClearChan
 * not used in gvs
 */
void pfClearChan( pfChannel *chan )
{
   chan;
}

/*  pfDraw
 * not used in gvs
 */
void pfDraw( void )
{
}


/* pfOverride
void pfOverride( long mask, long val )
{
   ( mask, val );
}

/* pfNotify
 *
 * pfNotify functions
 *  These functions provide a general purpose error message and
 *  notification handling facility for applications using IRIS
 *  Performer.This facility is used internally by IRIS Performer for
 *  error, warning, and status notifications and can be used by
 *  developed programs as well. No exception handling routines have been
 *  implemented for OpenGVS or PTG.These pfNotify series functions work
 *  fine with the following exception:pfNotify does not reformat output
 *  parameters, as you might expect it would. It just printf's the
 *  string. You'll need to get formatted output another way
 *  F.*/

/* ptgCheckNotifyEnvironment - utility to check/set env var
 *    if req. PTG_GL_NotifyThreshold can not be reset if PFNFYLEVEL
 *    has been set.
 * f - 1apr96
 */
```

```
void ptgCheckNotifyEnvironment( void )
{
#ifdef PFNFYLEVEL
   PTG_GL_PFNFYLEVEL_flag = 1;
   PTG_GL_NotifyThreshold = PFNFYLEVEL;
#endif
}


/*
 * pfNotifyHandler - set the global PTG_GL_DefaultHandlerFunc.
 * f - 1apr96
 */
extern void pfNotifyHandler( pfNotifyFuncType _handler )
{
   PTG_GL_DefaultHandlerFunc = _handler;
}


/*
 * pGetfNotifyHandler - return the set global PTG_GL_NotifyHandler,
 * f - 1apr96
 */
extern pfNotifyFuncType pfGetNotifyHandler( void )
{
   return PTG_GL_DefaultHandlerFunc;
}


/*
 * pfDefaultNotifyHandler - print a std msg in the form:
 *  PF<LEVEL>/<PFERROR>(<ERRNO>) <MESSAGE>.
 * f - 14apr96
 */
extern void pfDefaultNotifyHandler(pfNotifyData  *notice)
{
   /* Notice severity must be above the threshhold to be printed. */
   if (notice->severity < PTG_GL_NotifyThreshold) return;

   printf("PTG: pfNotify -> ");

   /* If a "more" error, just print msg. */
   if ( notice->pferrno == PFNFY_MORE )
   {
      printf("PF %s\n", notice->emsg);
      return;
   }

   switch ( notice->severity )
   {
      case (PFNFY_ALWAYS) :
```

```
        { printf("PF PFNFY_ALWAYS/");          break; }
      case (PFNFY_FATAL) :
        { printf("PF PFNFY_FATAL/");           break; }
      case (PFNFY_WARN) :
        { printf("PF PFNFY_WARN/");            break; }
      case (PFNFY_NOTICE) :
        { printf("PF PFNFY_NOTICE/");          break; }
      case (PFNFY_INFO) :
        { printf("PF PFNFY_INFO/");            break; }
      case (PFNFY_DEBUG) :
        { printf("PF PFNFY_DEBUG/");           break; }
      case (PFNFY_FP_DEBUG) :
        { printf("PF PFNFY_FP_DEBUG/");        break; }
      case (PFNFY_INTERNAL_DEBUG) :
        { printf("PF PFNFY_INTERNAL_DEBUG/"); break; }
      default: break;
   }
   switch ( notice->pferrno )
   {
      case (PFNFY_USAGE) :
        { printf("PFNFY_USAGE (%d) ",     notice->pferrno); break; }
      case (PFNFY_RESOURCE) :
        { printf("PFNFY_RESOURCE (%d) ",  notice->pferrno); break; }
      case (PFNFY_SYSERR) :
        { printf("PFNFY_SYSERR (%d) ",    notice->pferrno); break; }
      case (PFNFY_ASSERT) :
        { printf("PFNFY_ASSERT (%d) ",    notice->pferrno); break; }
      case (PFNFY_PRINT) :
        { printf("PFNFY_PRINT (%d) ",     notice->pferrno); break; }
      case (PFNFY_INTERNAL) :
        { printf("PFNFY_INTERNAL %d ",    notice->pferrno); break; }
      case (PFNFY_FP_OVERFLOW) :
        { printf("PFNFY_OVERFLOW (%d) ",  notice->pferrno); break; }
      case (PFNFY_FP_DIVZERO) :
        { printf("PFNFY_DIVZERO (%d) ",   notice->pferrno); break; }
      case (PFNFY_FP_INVALID) :
        { printf("PFNFY_INVALID (%d) ",   notice->pferrno); break; }
      case (PFNFY_FP_UNDERFLOW) :
        { printf("PFNFY_UNDERFLOW (%d) ", notice->pferrno); break; }
      default: break;
   }
   printf("%s\n", notice->emsg);

}


/*
 * pfNotifyLevel - set the global PTG_GL_NotifyThreshold,
 *   (if not prev set by env var)
 *   default is set to PTG_GL_NotifyData.severity = PFNFY_ALWAYS
 * f - 14apr96
```

```
 */
extern void pfNotifyLevel(int _severity)
{
   ptgCheckNotifyEnvironment();

   if( PTG_GL_PFNFYLEVEL_flag )
   {
      printf("PTG: pfNotifyLevel -> WARNING: level not reset.\n");
      printf("PTG: Use of env var PFNFYLEVEL disables further changes.
See man pages.\n");
      return;
   }

   printf("PTG: pfNotifyLevel -> Level reset.\n");
   PTG_GL_NotifyThreshold = _severity;
}


/*
 * pfGetNotifyLevel - returns the global PTG_GL_NotifyThreshold,
  *   Default is PTG_GL_NotifyThreshold = PFNFY_ALWAYS
 * f - 14apr96
 */
extern int pfGetNotifyLevel(void)
{
   ptgCheckNotifyEnvironment();
   return PTG_GL_NotifyThreshold;
}


/*
 * pfNotify - print error msg if
 *  set PTG_GL_NotifyData.severity >= input _severity.
 *  PFNFY_FATAL causes program exit.
 *  This function does not work as expected: there is no
 *  parameter parsing, and _format is simply printed. It
 *  can be fixed later, time permitting.
 * f - 14apr96
 */
extern void pfNotify(int _severity, int _error, char *_format, ...)
{
   ptgCheckNotifyEnvironment();

   PTG_GL_NotifyData.severity = _severity;
   PTG_GL_NotifyData.pferrno = _error;
   strcpy(PTG_GL_NotifyData.emsg, _format);

   PTG_GL_DefaultHandlerFunc(&PTG_GL_NotifyData);

   if( _severity == PFNFY_FATAL ) exit(1);
}
```

6. /Current/src/ptgLight.c

```
/*************************************************************
 * ptgLight.c
 * This file contains definitions and prototypes
 * of the Performer API which are part of the
 * project to port Performer to the mulit-platform
 * OpenGVS API. The Performer man pages and pr.c
 * code should be consulted for details.
 * This file contains the light specific source code.
 *************************************************************/

#include "../include/pfToGVS.h"


/* pfNewLSource - Performer and OpenGVS create
 *    Lights/LightSources identically.
 * f - 13mar96
 * j - 22feb96
 */
pfLightSource * pfNewLSource( void )
{
   static GV_Rgba ambient = { 0.0, 0.0, 0.0, 1.0 };
   static GV_Rgba diffuse = { 1.0, 1.0, 1.0, 1.0 };

   int ix;

   if ( PTG_GL_allocatedLsrs >= AVAIL_LSRS )
   {
      printf( "PTG: Fail pfNewLSource. Not enough AVAIL_LSRS.\n" );
      printf( "PTG: Usable pfLightSources 0 - %d. Exiting PTG.\n",
AVAIL_LSRS - 1 );
      exit( G_FAILURE );
   }

   GV_lsr_create( &PTG_GL_lsr[ PTG_GL_allocatedLsrs ] );

   /* performer initial states */

   if ( G_FAILURE ==
      GV_lsr_set_state  ( PTG_GL_lsr[ PTG_GL_allocatedLsrs ], G_ON) ||
      GV_lsr_set_ambient( PTG_GL_lsr[ PTG_GL_allocatedLsrs ],
                          &ambient) ||
      GV_lsr_set_diffuse( PTG_GL_lsr[ PTG_GL_allocatedLsrs ],
                          &diffuse)    )
         printf("PTG: pfNewLSource failure\n");
   else
      printf( "PTG: pfNewLSource -> created light is #%i \n",
               PTG_GL_allocatedLsrs);
```

```
   /* if hardware lighting turned on, light all scenes */
   if ( PTG_PFSTATE_ENLIGHTING )
   {
      for(ix=0; ix < PTG_GL_allocatedScns; ix++)
      {
         GV_scn_add_light(PTG_GL_scn[ix],
            PTG_GL_lsr[ PTG_GL_allocatedLsrs ]);
      }
   }
   return &PTG_GL_lsr[ PTG_GL_allocatedLsrs++ ];
}


/* pfNewLight
 * creates new light source
 *  j -t 22feb96
 */
pfLight* pfNewLight(void* _arena)
{
    _arena;
   return ( pfNewLSource() );
}


/* pfLightOn
 * turns light on
 * j -tested 21 feb96
 */
void pfLightOn(pfLight* _lt)
{
   int temp;

   temp = GV_lsr_set_state ( *_lt, G_ON );
   if (temp == G_FAILURE)
      printf("PTG: pfLightOn Failure\n");
}


/* pfLightModel
 * turns light on
 */
pfLightModel * pfNewLModel( void * vd )
{
 /* pfLight *light;
   int ix;

   light = pfNewLSource();

   for(ix=0; ix<PTG_GL_allocatedScns; ix++);
   {
    GV_scn_add_object  ( *( (GV_Scene*)PTG_GL_scn[ix]), *(GV_Obi*)light
```

```
);
   } */
      vd;   /* no direct use */
   return G_SUCCESS;
}


/* pfLightOff
 * turns light off
 * j -t 21 feb96
 */
void pfLightOff(pfLight* _lt)
{
   if (GV_lsr_set_state ( *_lt, G_OFF ) == G_FAILURE)
      printf("PTG: pfLightOff Failure\n");

}

/* pfIsLightOn
 * returns true if on, false if not
 *  j -t 22feb96
 */
long pfIsLightOn(pfLight* _lt)
{
   G_State state;
   GV_lsr_inq_state( *_lt, &state );
   if (state == G_ON )
      return TRUE;
   else
      return FALSE;
}


/*  pfLightPos --
 *  sets the light source position
 *  does not implement the w, local light
 *  gvs defaults to local light if set_position is used
 */
void pfLightPos(pfLight* lt, float x, float y, float z, float w)
{

   G_Position light_position;
   light_position.x  = x;
   light_position.y  = z;
   light_position.z = (-y);
    w;
   GV_lsr_set_position( *lt, &light_position);
}


/*  pfGetLightPos --
```

```
 *  gets the light source position
 *
 */
void pfGetLightPos(pfLight* lt, float* x, float* y, float* z, float* w)
{
   G_Position light_position;

   GV_lsr_inq_position( *lt, &light_position);

   *x =  (float)light_position.x;
   *y = -(float)light_position.z;
   *z =  (float)light_position.y;
   *w =  0.0;
}


/* pfApplyLModel
 *
 */
void pfApplyLModel( pfLightModel * lm )
{
   lm;
}


/*  pfLightAmbient
 *  sets the ambient color of the light
 *  default 1.0 for the alpha value
 *  j - t 22feb96
 */
void pfLightAmbient(pfLight* lt, float r, float g, float b)
{
   GV_Rgba ambient;

   ambient.r = r;
   ambient.g = g;
   ambient.b = b;
   ambient.a = 1.0;

   GV_lsr_set_ambient( *lt, &ambient );
}


/*  pfGetLightAmbient
 *  gets the ambient color of the light
 *  j -t 22feb96
 */
void pfGetLightAmbient( pfLight* lt, float* r, float* g, float* b )
{
   GV_Rgba ambient;
   GV_lsr_inq_ambient( *lt, &ambient );
```

```
    *r = ambient.r;
    *g = ambient.g;
    *b = ambient.b;
}


/* pfLightColor
 * performer uses color to specify the color of the light source
 * gvs uses diffuse to specify the color of the light source
 *  j -t 22feb96
 */
void pfLightColor(pfLight* lt, float r, float g, float b)
{
    GV_Rgba color;

    color.r = r;
    color.g = g;
    color.b = b;
    color.a = 1.0;
    GV_lsr_set_diffuse( *lt, &color );
}


/* pfGetLightColor
 * returns the current light color
 *  j -t 22feb96
 */
void pfGetLightColor(pfLight* lt, float* r, float* g, float* b)
{
    GV_Rgba color;

    GV_lsr_inq_diffuse(*lt, &color);

    *r = color.r;
    *g = color.g;
    *b = color.b;
}


/* pfSpotLightDir
 * sets a spot light to look in a specific direction
 *
 */
void pfSpotLightDir(pfLight* lt, float x, float y, float z)
{
    G_Vector3 direction;

    direction.x  = x;
    direction.y  = -z;
    direction.z  = y;
```

```
   GV_lsr_set_spot_direction(*lt, &direction );
}


/* pfGetSpotLightDir
 * returns the current spot light direction
 *
 */
void prGetSpotLightDir( pfLight* lt, float* x, float* y, float* z )
{
   G_Vector3 direction;

   GV_lsr_inq_spot_direction(*lt, &direction);

   *x = (float)  direction.x;
   *y = (float) -direction.z;
   *z = (float)  direction.y;
}


/* pgspotlightcone
 * set the exponent and cutoff of the spotlight
 *
 */
void  pfSpotLightCone( pfLight* lt, float f1, float f2 )
 {
    GV_lsr_set_spot_exponent( *lt,( f1 * G_DEG_TO_RAD ) );
    GV_lsr_set_spot_cutoff( *lt, ( f2 * G_DEG_TO_RAD ));
 }

/* pfgetspotlightcone
 * get the current settign of the spot light that is
 * passed in.  performer uses degrees, gvs uses radians
 *
 */
void  pfGetSpotLightCone( pfLight* lt, float* f1, float* f2 )
{
   GV_lsr_inq_spot_exponent( *lt, f1 );
   GV_lsr_inq_spot_cutoff( *lt, f2 );
   *f1 *= G_RAD_TO_DEG;
   *f2 *= G_RAD_TO_DEG;
}
```

7. /Current/src/ptgMath.c

```
/*************************************************************
 * ptgMath.c
 * NPS Performer to GVS Project function wrapper library
 * that contains the function for the conversion of pfmath
 * functions. The Performer man pages and prmath.c
 * code should be consulted for details.
 * This file contains the math specific source code.
 *************************************************************/
#include "../include/ptgMath.h"

/*  ptg_convert_PVec3_to_GVec3
 *  fuction converts float[3]
 *  to G_Vector3
 *  j 6 mar 96
 */
void  ptg_convert_PVec3_to_GVec3(const pfVec3 p_vector, G_Vector3
*g_vector)
{
   g_vector->x = p_vector[0];
   g_vector->y = p_vector[1];
   g_vector->z = p_vector[2];
}


/*  ptg_convert_GVec3_to_PVec3
 *  fuction converts  G_Vector3 to float[3]
 *
 * j 6 mar 96
 */
void  ptg_convert_GVec3_to_PVec3(G_Vector3 *g_vector, pfVec3 p_vector )
{
   p_vector[0] = g_vector->x;
   p_vector[1] = g_vector->y;
   p_vector[2] = g_vector->z;
}


/* pfSinCos - returns both sin/cos vals
 * f - 25jan96
 */
void pfSinCos(float angle, float * s, float * c)
{
   float temp = angle * G_DEG_TO_RAD;

   *c = cosf( temp );
   *s = sinf( temp );
}
```

```
float pfTan(float arg)
{
    return (tanf(arg));
}

float pfArcTan2(float y, float x)
{
   return (atan2f( y, x ));
}

float pfArcSin(float arg)
{
    return asinf( arg );
}

float pfArcCos(float arg)
{
    return acosf( arg );
}

float pfSqrt(float arg)
{
    return sqrtf( arg );
}

/* pfFPConfig
 * Performer uses this function to manipulate
 * the round-off ability in floating point
 * functions.  GVS does not support this.
 */
void pfFPConfig(long which, float val)
{
   (which, val);
}

/*
 * Performer uses this function to manipulate
 * the round-off ability in floating point
 * functions.  GVS does not support this.
 * Returns the default state only. Cannot
 * change these, just for compatibility.
 */
float pfGetFPConfig(long which)
{
   long conf;
   switch ( which )
   {
      case ( PFFP_UNIT_ROUNDOFF ):
      {
         conf = 1.0e-5;
      }
```

```
      case ( PFFP_ZERO_THRESH ):
      {
         conf = 1.0e-15;
      }
      case ( PFFP_TRAP_FPES ):
      {
         conf = 0.0;
      }
      default:
      {
         conf = -1.0;
      }
   }
   return conf;
}


void pfAddVec3(pfVec3 dst, const pfVec3 v1, const pfVec3 v2)
{
   PFADD_VEC3(dst, v1, v2);
}

/* pfCopyVec3 - copy from v to dst
 * f - t - 26 mar 96
 */
void pfCopyVec3(pfVec3 dst, const pfVec3 v)
{
   PFCOPY_VEC3( dst, v );
}

void pfScaleVec3(pfVec3 dst, float s, const pfVec3 v)
{
   G_Vector3 in_vector, out_vector;

   ptg_convert_PVec3_to_GVec3(v, &in_vector);
   G_vec_scale( s, &in_vector, &out_vector );
   ptg_convert_GVec3_to_PVec3(&out_vector, dst);
}


/* pfSetVec3 - assign values to V3 structs.
 * f t - 25jan96
 */
void pfSetVec3( pfVec3 dst, float x, float y, float z )
{
   PFSET_VEC3( dst, x, y, z );
}


void pfSubVec3(pfVec3 dst, const pfVec3 v1, const pfVec3 v2)
{
```

```
   PFSUB_VEC3(dst, v1, v2);
}


/* -------------------- Matrix -----------------------*/
void pfMultMat(pfMatrix dst, const pfMatrix m1, const pfMatrix m2)
{
   G_mat_mult_AxB_is_C( m1, m2, dst );
}


/*----------------- pfMaterial functions ---------------------------*/

/* create and initialize the new material
 * with performer values
 * j- 17FEB96
 */
pfMaterial * pfNewMtl( void * vd )
{
   static float transparency = 0.5;
   static float shininess    = 1.0;

   static GV_Rgba color = { 0.8, 0.5, 0.2, 1.0 };
   static GV_Rgba ambient   = { 0.2, 0.2, 0.2, 1.0 };
   static GV_Rgba diffuse   = { 0.8, 0.8, 0.8, 1.0 };
   static GV_Rgba emission  = { 0.0, 0.0, 0.0, 1.0 };
   static GV_Rgba specular  = { 0.0, 0.0, 0.0, 1.0 };
   vd;
   transparency;
   color;
  /* fail if not enough materials */
  if ( PTG_GL_allocatedMtls >= AVAIL_MTLS )
   {
      printf( "PTG: Fail pfFog. Not enough AVAIL_FOGS.\n" );
      printf( "PTG: Usable pfFog 0 - %d. Exiting PTG.\n", AVAIL_FOGS - 1
);
      exit( G_FAILURE );
   }

  printf("PTG: pfNewMtl  %i\n", PTG_GL_allocatedMtls);

  GV_mtl_create(&PTG_GL_material[ PTG_GL_allocatedMtls ]);
  GV_mtl_set_ambient(PTG_GL_material[ PTG_GL_allocatedMtls ], &ambient);
  GV_mtl_set_diffuse(PTG_GL_material[ PTG_GL_allocatedMtls ], &diffuse);
  GV_mtl_set_emission(PTG_GL_material[ PTG_GL_allocatedMtls ],
&emission);
  GV_mtl_set_specular(PTG_GL_material[ PTG_GL_allocatedMtls ],
&specular);
  GV_mtl_set_shininess(PTG_GL_material[ PTG_GL_allocatedMtls ],
shininess);
  GV_mtl_set_face(PTG_GL_material[ PTG_GL_allocatedMtls ],GL_FRONT);
```

```
  GV_mtl_define(PTG_GL_material[ PTG_GL_allocatedMtls ]);

  return  (pfMaterial*)&PTG_GL_material[ PTG_GL_allocatedMtls++ ];
}


/* pfMtlSide
 * Changes the side to which the material will
 * be applied.  Performer only allows these
 * three values.
 */
void  pfMtlSide( pfMaterial* mtl, long side )
{
   switch( side )
   {
      case ( PFMTL_FRONT ):
      {
         GV_mtl_set_face(*mtl ,GL_FRONT); break;
      }
      case ( PFMTL_BACK ) :
      {
         GV_mtl_set_face(*mtl ,GL_BACK); break;
      }
      case ( PFMTL_BOTH ) :
      {
         GV_mtl_set_face(*mtl ,GL_FRONT_AND_BACK);  break;
      }
      default :  { exit( G_FAILURE ); break; }
   }
}


/* pfGetMtlSide
 * Returns the side to which the material is
 * applied.
 */
long pfGetMtlSide(pfMaterial* mtl)
{
   GLenum side;
   long which;
   GV_mtl_inq_face(*mtl, &side);

   switch (side)
    {
       case( GL_FRONT)          : { which = PFMTL_FRONT; break; }
       case( GL_BACK)           : { which = PFMTL_BACK;  break; }
       case( GL_FRONT_AND_BACK): { which = PFMTL_BOTH;  break; }
   }
  return which;
}
```

```
/*  pfMtlAlpha
 *  sets the material alpha
 */
void pfMtlAlpha( pfMaterial * mtl, float alpha)
{
   static GV_Rgba color;

   GV_mtl_inq_ambient(*mtl, &color);
   if (alpha > 1.0) alpha = 1.0;

   if (color.a != alpha)
   {
      color.a = alpha;
      GV_mtl_set_ambient(*mtl, &color);

      GV_mtl_inq_diffuse(*mtl, &color);
      color.a = alpha;
      GV_mtl_set_diffuse(*mtl, &color);

      GV_mtl_inq_emission(*mtl, &color);
      color.a = alpha;
      GV_mtl_set_emission(*mtl, &color);

      GV_mtl_inq_specular(*mtl, &color);
      color.a = alpha;
      GV_mtl_set_specular(*mtl, &color);
   }
}

/*
 *  returns the material alpha
 */
float  pfGetMtlAlpha(pfMaterial* mtl)
{
  static GV_Rgba color;

  GV_mtl_inq_ambient(*mtl, &color);

  return color.a;
}

/*
 *  sets the material shininess
 *  performer uses 0.0 to 1.0
 */
void pfMtlShininess( pfMaterial * mtl, float shininess)
{
   if (shininess > 1.0) shininess = 1.0;
```

```
   GV_mtl_set_shininess(*mtl, shininess);
}


/*  pfGetMtlShininess
 *  returns the material shininess
 *  performer uses 0.0 to 1.0
 */
float  pfGetMtlShininess(pfMaterial* mtl)
{
  float shininess;

  GV_mtl_inq_shininess(*mtl, &shininess);
  return shininess;
}


/* pfMtlColor
 * sets the material color of (ambient, diffuse, emmissive, specular)
 *
 */
void pfMtlColor( pfMaterial * mtl, long color, float r, float g, float b
)
{
   static GV_Rgba newcolor;

   GV_mtl_inq_ambient(*mtl, &newcolor);


   if (r >1.0) r = 1.0;
   if (b >1.0) b = 1.0;
   if (g >1.0) g = 1.0;

   newcolor.r = r;
   newcolor.g = g;
   newcolor.b = b;

   switch ( color )
   {
      case ( DIFFUSE )  :
      {
         GV_mtl_set_diffuse(*mtl, &newcolor);
         break;
      }
      case ( AMBIENT )  :
      {
         GV_mtl_set_ambient(*mtl, &newcolor);
         break;
      }
      case ( EMISSION ) :
      {
```

```
            GV_mtl_set_emission(*mtl, &newcolor);
            break;
         }
         case ( SPECULAR ) :
         {
            GV_mtl_set_specular(*mtl, &newcolor);
            break;
         }
      }
   }


/* pfGetMtlColor
 * returns the material color of acolor
 *
 */
void  pfGetMtlColor( pfMaterial* mtl, long acolor, float* r, float* g,
float* b )
{
   static GV_Rgba color;


   switch ( acolor )
   {
      case ( DIFFUSE )  :
      {
         GV_mtl_inq_specular(*mtl, &color);
         break;
      }

      case ( AMBIENT )  :
      {
         GV_mtl_inq_specular(*mtl, &color);
         break;
      }
      case ( EMISSION ) :
      {
         GV_mtl_inq_specular(*mtl, &color);
         break;
      }
      case ( SPECULAR ) :
      {
         GV_mtl_inq_specular(*mtl, &color);
         break;
      }
      default : break;
   }

   *r = color.r;
   *g = color.g;
```

```
   *b = color.b;
}


/*   pfApplyMtl
 *
*/
void pfApplyMtl( pfMaterial * mtl )
{
   GV_mtl_define(*mtl);
   GV_mtl_set_current(*mtl);
}
void PTG_apply_geostate(pfGeoState *geostate)
{

   GV_mtl_define(geostate->frontmtl);
   GV_mtl_set_current(geostate->frontmtl);
}
```

8. /Current/src/ptgSmoke.c

```
/*****************************************************************
*
*  ptgSmoke.c --
*
*  NPS Performer to GVS Project function wrapper library
*  that contains the function for the conversion of pfuSmoke
*  functions. Because of the differences in which gvs & performer create
*  and modify the smoke model, some of the fuctions were given defaults
*  and will not be able to be changed.  The affected functions are
*  mentioned below.
*****************************************************************/

#include "../include/pfToGVS.h"

/* pfuNewSmoke
 * creates a new smoke and sets up initial
 * defaults of the type.
 * j t
 */
pfuSmoke* pfuNewSmoke(void)
{
   if ( PTG_GL_allocatedSmoke >= AVAIL_SMOKES )
   {
      printf( "PTG: Fail pfSmoke. Not enough AVAIL_SMOKES.\n" );
      printf( "PTG: Usable pfSmoke 0 - %d. Exiting PTG.\n", AVAIL_SMOKES
- 1 );
      exit( G_FAILURE );
   }

   /* create the GVS smoke */
   GVU_smk_create( &PTG_GL_Smoke[ PTG_GL_allocatedSmoke ] );
   printf("PTG: pfNewSmoke -> %d \n",PTG_GL_allocatedSmoke );

 /* apply the smoke to the scenes
  * smoke is applied at least once and at most twice to scene
  * to ensure that smoke created dynamically is applied to the scene
  * smoke created prior to sim loop will be applied here if possible
  * and in post main.  this is to ensure that if a the smoke is created
  * before the scene, the smoke will still be applied.
  */
   PTG_apply_smoke;

   /* return the address of the handle */
   return (&PTG_GL_Smoke[ PTG_GL_allocatedSmoke++]);
}


/* pfuSmokeType
```

```
 * Determines what type of "smoke" will be displayed.
 * Performer uses smoke as fire, in gvs we will display
 * it as smoke
 * smoke/dust/explosion/missile will have different smoke
 * characteristics.
 *
 * j t
 */
void  pfuSmokeType(pfuSmoke *smoke, long type_of_smoke)
{
   /* choose to have 5 ramps for directions/colors/times        */
   /* gvs allows a direction ramp, color ramp & velocity ramp   */
   /* that changes the smoke according to where in the ramp     */
   /* the smoke is.                                             */

   float density;              /* thickness of smoke, not            */
                               /* smoke puffs/frame (gvs)            */
   float fade_time,            /* time for smoke to dissapate        */
         color_times[5],       /* array of times to apply color      */
         direction_times[5];   /* array of times to apply directions */
   int ncolors,                /* number of colors                   */
       ndirections;            /* number of directions               */

   G_Vector3 directions[5];    /* direction of smoke */
   GV_Rgba colors[5];          /* color of smoke   */


   /* change the above states based upon the type of smoke we */
   /* want to create   */
   switch ( type_of_smoke )
   {
      case ( PFUSMOKE_MISSLE ):
      {
         break;
      }
      case ( PFUSMOKE_EXPLOSION ):
      {
         break;
      }
      case ( PFUSMOKE_FIRE ):
      {
         density = 1.0;
         fade_time = 2.0;
         ndirections = 1;
         ncolors = 3;

         /* the start time to apply directions[X] */
         direction_times[0] = 0.0 ;

         /* initial direction */
         directions[0].x = 0.0;
```

```
    directions[0].y = 1.0;
    directions[0].z = 0.0;

    /* the start time to change the color to color[X] */
    color_times[0] = 0.0;
    color_times[1] = 0.3;
    color_times[2] = 0.6;

    /* color states   a redish fire */
    colors[0].r = 1.0;
    colors[0].g = 0.3;
    colors[0].b = 0.0;
    colors[0].a = 0.5;

    colors[1].r = 1.0;
    colors[1].g = 0.3;
    colors[1].b = 0.0;
    colors[1].a = 0.7;

    colors[2].r = 1.0;
    colors[2].g = 0.3;
    colors[2].b = 0.0;
    colors[2].a = 0.9;


    break;
}
default :
/*case ( PFUSMOKE_SMOKE ):
  produces a thick dark smoke */
{
    density = 1.0;
    fade_time = 15.0;
    ndirections = 2;
    ncolors = 3;

    /* the start times to apply directions[X] */
    direction_times[0] = 0.0 ;
    direction_times[1] = 10.0 ;

    /* initial direction */
    directions[0].x = 0.0;
    directions[0].y = 1.0;
    directions[0].z = 0.0;

    directions[1].x = 0.2;
    directions[1].y = 1.0;
    directions[1].z = 0.1;

    /* the start time to change the color to color[X] */
    color_times[0] = 0.0;
```

```
            color_times[1] = 1.0;
            color_times[2] = 10.0;

            /* color states  thick dark smoke  */
            colors[0].r = 0.1;
            colors[0].g = 0.1;
            colors[0].b = 0.1;
            colors[0].a = 0.3;

            colors[1].r = 0.1;
            colors[1].g = 0.1;
            colors[1].b = 0.1;
            colors[1].a = 0.8;

            colors[2].r = 0.15;
            colors[2].g = 0.15;
            colors[2].b = 0.15;
            colors[2].a = 0.3;

          break;
        }
        case ( PFUSMOKE_DUST ):
        {

           break;
        }
    }
          /* apply density */
          GVU_smk_set_density( *smoke, density );

          /* apply fade time */
          GVU_smk_set_fade_time( *smoke, fade_time );

           /* set the smoke directions */
           GVU_smk_set_direction_vectors( *smoke, ndirections,
               direction_times, directions );

           /* set the smoke colors */
           GVU_smk_set_color_ramp( *smoke, ncolors, color_times, colors );

}


/* pfuInitSmokes
 * Not used in GVS
 */
void pfuInitSmokes(void)
{

}
```

```
/*  pfuSmokeDuration
 *  Used to set the duration of the smoke
 *  j
 */
void pfuSmokeDuration(pfuSmoke* smoke, float dur)
{
   GVU_smk_set_duration(*smoke, dur );
}



/* pfuSmokeColor
 * performer sets the beginning and ending color of a smoke
 * gvs uses a color ramp
 * Will attempt to simulate this be finding the delta of
 * each of the colors (rgba) and using a 7 color ramp in
 * GVS. Will also take the delta time of the fade time.
 * j t.
 */
void pfuSmokeColor(pfuSmoke* smoke, pfVec3 bgn, pfVec3 end)
{
   float  delta_red, delta_green, delta_blue;
   float time_out[10];

   /* choose 10 */
   GV_Rgba color_in[10], color_out[10];
   int number_of_points = 10;
   int number_of_points_used;
   int ix;


   GVU_smk_inq_color_ramp(*smoke, number_of_points, time_out,
            color_out, &number_of_points_used);

   /* find the delta of red */
   delta_red = (bgn[0] + end[0])/number_of_points_used;

   /* find the delta of green */
   delta_green = (bgn[1] + end[1])/number_of_points_used;

   /* find the delta of blue */
   delta_blue = (bgn[2] + end[2])/number_of_points_used;

   /* set the initial condition */
   color_in[0].r = bgn[0];
   color_in[0].g = bgn[1];
   color_in[0].b = bgn[2];
   color_in[0].a = color_out[0].a;

   /* set the color ramp */
```

```
   for (ix=1; ix <= number_of_points_used; ix++)
   {
      color_in[ix].r = color_in[ix-1].r + delta_red;
      color_in[ix].g = color_in[ix-1].g + delta_green;
      color_in[ix].b = color_in[ix-1].b + delta_blue;
      color_in[ix].a = color_out[ix-1].a;
   }

   /* create the color ramp */
   GVU_smk_set_color_ramp(*smoke, number_of_points_used, time_out,
color_in);
}



/*   pfuSmokeDensity
 *
 *   sets the density of the smoke (number of puffs per frame )
 *   choose not to allow modification to the density to ensure
 *   realistic smoke.  gvs does not need this to achive good smoke.
 *   j
 */
void  pfuSmokeDensity(pfuSmoke* smoke, float dens, float diss, float
expansion)
{
   (smoke, dens, diss, expansion );
}


/*    pfuSmokeVelocity
 *
 *    adjusts the velocity of the smoke.  Performer does not utilize the
 *    gvs type velocities at different times.  because of this, the
 *    translation did not appear correct, so disabled the modification
 *    of the velocity of the smoke.
 *    j
 */
void  pfuSmokeVelocity(pfuSmoke* smoke, float turbulence, float speed)
{
  (smoke, turbulence, speed );
}


/*   pfuGetSmokeVelocity
 *   returns the smoke velocity
 *   j
 */
void  pfuGetSmokeVelocity(pfuSmoke* smoke, float *turbulence, float
*speed)
{
   int    npoints = 5,
```

```
          npoints_used;

   float  velocity[5];
   float  time_out[5];
   turbulence;
   GVU_smk_inq_velocity_ramp( *smoke, npoints, time_out, velocity,
&npoints_used);

   *speed = velocity[0];
}


/* pfuSmokeDir
 * Sets the smoke direction.  may want to add random directions as time
 * increases.
 * j
 */
void pfuSmokeDir(pfuSmoke* smoke, pfVec3 dir)
{

   float time_in[5] = {0.0};
   int number_in  = 5;
   G_Vector3 direction_in[5];

   direction_in[0].x = dir[0];
   direction_in[0].y = -dir[2];
   direction_in[0].z = dir[1];

   GVU_smk_set_direction_vectors(*smoke, number_in, time_in,
direction_in);
}


/*  pfuGetSmokeDensity
 *  returns the density
 */
void  pfuGetSmokeDensity(pfuSmoke* smoke, float *dens, float *diss,
float *expansion)
{
     expansion;
     GVU_smk_inq_density(*smoke,dens);
     GVU_smk_inq_fade_time(*smoke, diss );
}


/* pfuSmokeMode
 *
 * turns the smoke on and off based on the mode
 * j t
 */
 void pfuSmokeMode(pfuSmoke* smoke, long mode)
```

```
{

   switch ( mode )
   {
      case ( PFUSMOKE_START ) :
      {
         GVU_smk_start( *smoke );
         break;
      }
      case ( PFUSMOKE_STOP ) :
      {
         GVU_smk_stop( *smoke );
         break;
      }
      default: break;
   }
}


/*  pfuDrawSmokes
 *  not used in gvs
 *  j
 */
void pfuDrawSmokes(pfVec3 eye) /* Draw Process only */
{
   (eye);
}



/* pfuSmokeOrigin
 *
 * Sets the origin of the smoke
 * when settting the smoke size according to radius, smoke did
 * not appear very good, so the size was disabled.
 * j t
 */
void pfuSmokeOrigin(pfuSmoke* smoke, pfVec3 origin, float radius)
{
    G_Position position;
    radius;
    position.x =  origin[0];
    position.y = -origin[2];
    position.z =  origin[1];

    /* set the smoke at the location */
    GV_obi_set_position( *smoke, &position );
}


/*  PTG_apply_smoke
```

```
 *  associates the smoke model to a channel if available
 *  if channel is not available, (smoke is created & applied
 *  prior to a channel being created) the smoke is applied
 *  in the pfPostMain fuction.
 *  j t
 */
void PTG_apply_smoke(void )
{
   int ix, jx = 0;

   /* step through smokes created */
   for (ix = 0; ix < PTG_GL_allocatedScns; ix++)
   {
      for (jx=0; jx < PTG_GL_allocatedSmoke; jx++)
      {  /* step through the scenes created */
         if ( PTG_GL_scn[ ix ] != 0  &&  PTG_GL_Smoke[ jx ] != 0)
         {
            GV_scn_add_object( PTG_GL_scn[ ix ] , PTG_GL_Smoke [ jx ] );
         }
      }
    }

}
```

9. /Current/src/ptgUtils.c

```
/*****************************************************************************
*
*  ptgUtils.c --
*
*       NPS Performer to OpenGVS Project Utilities
*
*****************************************************************************/


#include "../include/pfToGVS.h"

/*
 * GV_user_init - automatically called once during system
 * initialization.Do pre-pfMain, pfMain, post-pfMain. After this, all
 * sim loop setup should be complete.
 * f - 1feb96
 */
int GV_user_init( void )
{
    printf("PTG: Performing GV_user_init()\n");

    /* Call pre-pfMain function. Whatever setup work is required. */
    PTG_pre_pfMain();

    printf("PTG: Performing pfMain()\n");
    pfMain( PTG_GL_argc, PTG_GL_argv );

    /* Call post-pfMain function. Last minute stuff before sim loop. */
    PTG_post_pfMain();

    return G_SUCCESS;
}



/*
 *
 *  GV_demo_sys --
 *
 *       Establish callbacks to appropriate project specific routines
 *       and initialize GVS for Performer port.
 *  f - 3feb96
 */
int GV_demo_sys( int argc, char ** argv )
{
    GV_Sys_mode system_mode;

    printf("PTG: Performing GV_demo_sys()\n");
```

```
    /*  Use Meters, just like in Performer for easier port. */
    G_sys_set_units( 1.0, G_SYS_UNITS_METERS );

    /* User routine to parse the Unix command line */
    /* GV_user_parse_cmd( argc, argv );  */

    PTG_GL_argc = argc;
    PTG_GL_argv = argv;

    /* Establish project specific callbacks */
    GV_sys_set_callback_init( GV_user_init );
    GV_sys_set_callback_proc( GV_user_proc );
    GV_sys_set_callback_shutdown( (GV_sys_callback)PTG_GlobalClose );

    /* Initialize OpenGVS */
    printf( "PTG: Performing GV_sys_init(). [All callbacks have been
             set.] \n");
    GV_sys_init();

    GV_sys_set_mode( GV_SYS_MODE_RUNTIME );
    GV_sys_inq_mode( &system_mode );


    printf("PTG: Begin simulation looping in GV_user_proc...\n");
    while (system_mode == GV_SYS_MODE_RUNTIME)
    {
 /* Loop here "forever" */
 GV_sys_proc();
 GV_sys_inq_mode( &system_mode );
    }

    /* Shutdown OpenGVS if not already done by user */
    if (system_mode == GV_SYS_MODE_SHUTDOWN)
 GV_sys_shutdown();

    return (G_SUCCESS);
}



/*
 *  drawnull - produces a null definition to go with
 *     new pfGroups, pfDCSs, ...
 *     Without it, a new GV_Obi can not be created.
 *     This is a little slight of hand required because Performer
 *     specifically "new"s DCSs, etc, but OpenGVS does not
 *     create the GV_Obi resource until it has an Obd assigned.
 *     This requires PTG to ensure an Obd is assigned prior to
 *     creating the object tree. So, this is a small object to do
 *     that. It's only 1 small poly, and it won't ever be seen.
```

```
 *     f - 3feb96
 */
void drawnull( void )
{
   float halfside = 0.5;
   float p[4][3];
   float x = 0.0;
   float y = 0.0;
   float z = 0.0;
   /* face */
   p[0][0]=x-halfside; p[0][1]=y+halfside; p[0][2]=z-halfside;
   p[1][0]=x+halfside; p[1][1]=y+halfside; p[1][2]=z-halfside;
   p[2][0]=x+halfside; p[2][1]=y-halfside; p[2][2]=z-halfside;
   p[3][0]=x-halfside; p[3][1]=y-halfside; p[3][2]=z-halfside;
   glColor3f(0.0, 0.0, 0.0);
   glBegin(GL_QUADS);
      glVertex3fv(p[0]);
      glVertex3fv(p[1]);
      glVertex3fv(p[2]);
      glVertex3fv(p[3]);
   glEnd();
}



/* Build_the_null_def - assists in creating new Obi's
 *    that do not have Obd's yet assigned. See "drawnull".
 *  f - 3feb96
 */
GV_Obd build_the_null_def( void )
{
    GV_Obd definition;
    GV_obd_open_by_name( "NULLDEF", &definition);
    {
    drawnull();
    }
    GV_obd_close( definition );
    return( definition );
}



/*
 * ptgGlobalInit
 *    Set global var defaults. Called by main() in gv_main.c .
 * f - t - 25Jan96
 */
void PTG_GlobalInit ( void )
{
   int ix = 0;
```

```
/* init arg holders */
PTG_GL_argc = 0;
PTG_GL_argv = 0;

/* print init info banner */
printf("PTG: NPS Performer to OpenGVS Project\n");
printf("PTG:   ver: %s  current: %s\n", PTG_GL_ver, PTG_GL_date);
printf("PTG: Performing PTG_GlobalInit()\n");

/*  init std 1/4 OpenGVS screen  */
PTG_GL_normalized_viewport.xmin = 0.0;
PTG_GL_normalized_viewport.xmax = 1.0;
PTG_GL_normalized_viewport.ymin = 0.0;
PTG_GL_normalized_viewport.ymax = 1.0;

/*  init std clip planes/aov  */
PTG_GL_yon = 10000.0f;
PTG_GL_hither = 1.0;
PTG_GL_aov = 45.0;

/* init simulation time */
PTG_GL_sim_time = 0.0f;

/* init camera platform
 *  capabilities with mult cameras per channel are unexploited
 *  by Performer - so should not probably be changed.
 */
PTG_GL_currentPlatform = 0;

/* init camera pos/rot */
PTG_GL_campos.x = 0.0f;
PTG_GL_campos.y = 0.0f;
PTG_GL_campos.z = 100.0f;
PTG_GL_camrot.x = 0.0f;
PTG_GL_camrot.y = 0.0f;
PTG_GL_camrot.z = 0.0f;

/* Current pfNotifyLevel - Can not be reset during
 * run if PFNFYLEVEL is used. See man pages.
 */
PTG_GL_PFNFYLEVEL_flag = 0;
PTG_GL_NotifyThreshold = PFNFY_ALWAYS;
PTG_GL_NotifyData.severity = 0;
PTG_GL_NotifyData.pferrno = 0;
PTG_GL_NotifyData.emsg = (char*)(G_malloc( sizeof(G_Name) ));
PTG_GL_NotifyData.emsg[ 0 ] = 0;
PTG_GL_DefaultHandlerFunc = pfDefaultNotifyHandler;

/* init erase color */
PTG_GL_erase_color.r = 0.0f;
PTG_GL_erase_color.g = 0.0f;
```

```
    PTG_GL_erase_color.b = 0.0f;
    PTG_GL_erase_color.a = 1.0f;

    /* init window title name */
    strncpy( PTG_GL_initName, "NPS Performer to OpenGVS Project\0",
             G_NAME_LENGTH );
    PTG_GL_initName[ G_NAME_LENGTH ] = 0;

    /* init import object file path */
    for ( ix = 0; ix < AVAIL_PATHS; ix++ )
    {
       PTG_GL_filePath[ ix ][ 0 ] = PTG_GL_filePath[ ix ]
          [ G_NAME_LENGTH ] =
          PTG_GL_filePathLength[ix] = 0;
    }
    PTG_GL_pathNumber = -1;
    for ( ix = 0; ix < AVAIL_FILES; ix++ )
       PTG_GL_importCmd[ ix ][ 0 ] = PTG_GL_importCmd[ ix ][
G_NAME_LENGTH ] = 0;
    PTG_GL_fileNumber = 0;
    PTG_GL_filePathListSet[ 0 ] = PTG_GL_filePathListSet[ G_NAME_LENGTH ]
= 0;

    /* init resources for allocation */
    PTG_GL_allocatedFbfs = 0L;
    PTG_GL_allocatedChns = 0L;
    PTG_GL_allocatedCams = 0L;
    PTG_GL_allocatedScns = 0L;
    PTG_GL_allocatedLsrs = 0L;
    PTG_GL_allocatedObis = 0L;
    PTG_GL_allocatedNodes = 0L;
    PTG_GL_allocatedESky = 0L;
    PTG_GL_allocatedFog  = 1L; /* fog[0] is reserved for current fog. */
    PTG_GL_allocatedGeoStates = 0L;
    PTG_GL_allocatedMtls      = 0L;
    PTG_GL_allocatedSmoke     = 0L;

    for (ix=0; ix<AVAIL_FBFS; ix++) PTG_GL_fbf[ix]=0;
    for (ix=0; ix<AVAIL_CHNS; ix++) PTG_GL_chn[ix]=0;
    for (ix=0; ix<AVAIL_CAMS; ix++) PTG_GL_cam[ix]=0;
    for (ix=0; ix<AVAIL_SCNS; ix++) PTG_GL_scn[ix]=0;
    for (ix=0; ix<AVAIL_LSRS; ix++) PTG_GL_lsr[ix]=0;
    for (ix=0; ix<AVAIL_OBIS; ix++) PTG_GL_obi[ix]=0;
    for (ix=0; ix<AVAIL_NODES; ix++) PTG_GL_node[ix]=0;
    for (ix=0; ix<AVAIL_EARTHSKYS; ix++) PTG_GL_ESky[ix].assoc_channel=0;

    /* build null obd to facilitate Performer tree building */
    GV_obd_inq_by_name( "NULLDEF", &PTG_GL_null_obd );
    PTG_GL_null_obd = build_the_null_def();
```

```
/*****************************************************************/
for (ix=0; ix<AVAIL_EARTHSKYS; ix++)
{
   /* mode is created to store the values for the two types of
      modes that can be stored in an earthsky model.  the mode[0]
      is used to store the value for PFES_BUFFER_CLEAR and mode[1] is
      used to store the values for PFES_CLOUD which only accepts one
      value (PFES_OVERCAST)
   */

  /* set the modes to defaults (performer) (valid range from 0 up) */
   PTG_GL_ESky[ix].mode[0] = PFES_FAST;
   PTG_GL_ESky[ ix ].mode[1] = PFES_OVERCAST;


   /* performer has defaults for the following items.  */
   /* set all the color to these defaults */
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_SKY_TOP,    0.0, 0.0, 0.44,
                0.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_SKY_BOT,    0.0, 0.4, 0.7,
                0.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_HORIZ,      0.8, 0.8, 0.1,
                0.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_GRND_FAR,   0.4, 0.2,0.0,
                1.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_GRND_NEAR, 0.5, 0.3, 0.0,
                1.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_CLOUD_TOP, 0.8, 0.8, 0.8,
                0.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_CLOUD_BOT, 0.8, 0.8, 0.8,
                0.0 );
   pfESkyColor( &PTG_GL_ESky[ ix ], PFES_CLEAR,      0.0, 0.0, 0.0,
                0.0 );



   /* set to Performer default states.  since the attribute values
      (such as PFES_GRND_HT) are defined from 310-318 and the array
      is defined from 0-8, I choose to simply remove the base(310)
      the values & only use the offset to allow for storage of the
      values.
   */

   /* read [attribute - base]        */
   PTG_GL_ESky[ ix ].attrib[ PFES_GRND_HT - PFES_CLOUD_TOP ]
      = 0.0;
   PTG_GL_ESky[ ix ].attrib[ PFES_HORIZ_ANGLE - PFES_CLOUD_TOP ]
      = 10.0;
   PTG_GL_ESky[ ix ].attrib[ PFES_CLOUD_TOP - PFES_CLOUD_TOP ]
      = 20000.0;
   PTG_GL_ESky[ ix ].attrib[ PFES_CLOUD_BOT - PFES_CLOUD_TOP ]
```

```
        = 20000.0;
    PTG_GL_ESky[ ix ].attrib[ PFES_TZONE_TOP - PFES_CLOUD_TOP ]
        = 25000.0;
    PTG_GL_ESky[ ix ].attrib[ PFES_TZONE_BOT - PFES_CLOUD_TOP ]
        = 15000.0;
    PTG_GL_ESky[ ix ].attrib[ PFES_GRND_FOG_TOP - PFES_CLOUD_TOP ]
        = 0.0;

    /* array locations 4 & 5 of attrib are not used, so we will use
       them for PFES_GRND(380) & PFES_GENERAL(381) which are also
       attributes, but are  not in the same range (310-318) as the
       other attribues.  So to get 380 & 381 to "fit"into attrib[4] &
       [5], I choose to use 376 as a base in order to use the[attrib-
       base]
    */

    PTG_GL_ESky[ ix ].attrib[ PFES_GRND - 376 ] =0;
    PTG_GL_ESky[ ix ].attrib[ PFES_GENERAL - 376 ] = 0;

    /* ground the pfchannel that is in every earthsky model */
    PTG_GL_ESky[ ix ].assoc_channel = NULL;

}


/* global states for the geostates modes*/
PTG_PFSTATE_TRANSPARENCY = PF_OFF;
PTG_PFSTATE_ANTIALIAS    = PF_OFF;
PTG_PFSTATE_DECAL        = PF_OFF;
PTG_PFSTATE_CULLFACE     = PF_OFF;
PTG_PFSTATE_ALPHAFUNC    = PF_OFF;

PTG_PFSTATE_ALPHAREF     =  0;

PTG_PFSTATE_ENLIGHTING   = PF_ON;
PTG_PFSTATE_ENTEXTURE    = PF_ON;
PTG_PFSTATE_ENFOG        = PF_ON;

PTG_PFSTATE_ENWIREFRAME     = PF_OFF;
PTG_PFSTATE_ENCOLORTABLE    = PF_OFF;
PTG_PFSTATE_ENHIGHLIGHTING  = PF_OFF;


/* global states for the geostates attributes*/
/* PTG_PFSTATE_LIGHMODEL  = NULL;
 PTG_PFSTATE_LIGHTS       = NULL;
 PTG_PFSTATE_TEXENV       = NULL;
 PTG_PFSTATE_TEXTURE      = NULL;
 PTG_PFSTATE_FOG          = NULL;
 PTG_PFSTATE_COLORTABLE   = NULL;
 PTG_PFSTATE_HIGHLIGHTING = NULL;
```

```
   */

    PTG_GL_Phase_mode = PFPHASE_FREE_RUN;
    PTG_GL_Frame_rate = -1.0;

    /* unused by PTG placeholders */
    PTG_GL_MultiProcessMode = PFMP_DEFAULT;
    PTG_GL_MultiPipeMode = 1;


}


/*
 * PTG_pre_pfMain
 *   Complete any pre-pfMain processing to get ready.
 * f - t - 1Feb96
 */
void PTG_pre_pfMain( void )
{
   printf("PTG: Performing PTG_pre_pfMain()\n");
}


/*
 * PTG__resource_pointer_lookup
 *   Check the resource list & find out what kind of pointer
 *   we have as an arg. This is used by Performer calls which
 *   use nodes (and their relatives) as args, but we need to
 *   use specific calls in OpenGVS based on resource type.
 * f - t - 14Feb96
 */
PTG_pointer_type PTG_resource_pointer_lookup( void* in_pointer)
{
    int ix = 0;
    int pointerFound = 0;
    PTG_pointer_type in_pointer_type = unknown_type;
    printf( "PTG:    PTG_resource_pointer_lookup().\n" );
    /* check scene lookup */
    for( ix = 0; ix < PTG_GL_allocatedScns; ix++)
    {
   if( ( (GV_Scene*)in_pointer ) == &PTG_GL_scn[ ix ] )
   {
      pointerFound = 1;
      in_pointer_type = scene_type;
      if( pointerFound )
             {
               printf( "PTG:    pfScene node found\n" );
               return in_pointer_type;  /* leave scene lookup loop */
             }
   }
```

```
  }

  /* check obi-group-dcs-scs lookup */
  for( ix = 0; ix < PTG_GL_allocatedObis; ix++)
  {
 if( ( (GV_Obi*)in_pointer ) == &PTG_GL_obi[ ix ] )
 {
    pointerFound = 1;
    in_pointer_type = obinstance_type;
    if( pointerFound )
           {
             printf( "PTG:    obi-group-dcs-scs node found\n" );
             return in_pointer_type;  /* leave obi lookup loop */
           }
 }
  }

 for( ix = 0; ix < PTG_GL_allocatedLsrs; ix++)
 {
       if( ( (GV_Light*)in_pointer ) == &PTG_GL_lsr[ ix ] )
{
   pointerFound = 1;
   in_pointer_type = light_type;
   if( pointerFound )
          {
             printf( "PTG:    pfLightSource node found\n" );
             return in_pointer_type;  /* leave light lookup loop */
          }
}
  }


/* check obi/pfNode lookup */
 for( ix = 0; ix < PTG_GL_allocatedNodes; ix++)
 {
    if( ( (GV_Obi*)in_pointer ) == &PTG_GL_node[ ix ] )
    {
       pointerFound = 1;
       in_pointer_type = node_type;
       if( pointerFound )
       {
          printf( "PTG:    GV_Obi/Node found\n" );
          return in_pointer_type;  /* leave node lookup loop */
       }
    }
 }

 /* put other possible child type searches here */

 /* if all else fails... */
 printf( "PTG:    Pointer lookup fails. Unknown returned.\n" );
```

```
   return unknown_type;
}



/*
 * PTG_post_pfMain
 *   Perform any post-pfMain processing before sim loop.
 * f - t - 1Feb96
 */
void PTG_post_pfMain( void )
{
   printf("PTG: Performing PTG_post_pfMain()\n");
   PTG_apply_fog();
   PTG_apply_smoke();
}



/*
 * PTG_GlobalClose
 *    cleanup & exit GVS after sim looping is done.
 * f - t - 25Jan96
 */
void PTG_GlobalClose ( void )
{
   GV_user_shutdown();
   printf("PTG: Exiting PTG \n");
}


     /*Compute new view position.*/
     si_t = pfGetTime();
     pfSinCos(45.0f*si_t, &s, &c);
     pfSetVec3(view.hpr, 45.0f*si_t, -10.0f, 0);
     pfSetVec3(view.xyz, 100.0f*s, -110.0f*c, 30.0f);
     pfChanView(si_chan, view.xyz, view.hpr);

     /*Initiate cull/draw for this frame.*/
     pfFrame();

    return G_SUCCESS;
}
```

APPENDIX C
PERFORMER TO OPENGVS EXAMPLE PROGRAMS

This appendix contains the PTG demo programs in both the original Performer version and the "ported" OpenGVS/PTG version. The purpose of both is to be able to compare the Performer demos with their OpenGVS/PTG translations. These programs are meant to demonstrate the nearness of the PTG translation library executables to the Performer originals, with the advantage of the former being the multi-platform capabilities of OpenGVS.

The Performer demos compile using the SGI Performer engine. You will need Performer 1.2 to compile and run the Performer directory demos. The OpenGVS demos compile using PTG wrapper for their Performer calls, and the multi-platform OpenGVS engine for graphics. The OpenGVS were last built with OpenGVS V4.0-b15.

Each demo section contains: the original Performer demo, OpenGVS/PTG ported demo, and the necessary source code (**`ptgprog.c`** and **`ptgprog.h`**) for each PTG demo.

The following programs are contained in this appendix:

a. **`simple.c`**
b. **`inherit.c`**
c. **`multichan.c`**
d. **`multipipe.c`**
e. **`earthsky.c`**
f. **`fog.c`**
g. **`smoke.c`**

A. SIMPLE DEMO

1. \PTG\examples\Performer\simple.c

```
/*
 * simple.c: simple Performer program for programmer's guide
 *
 * $Revision: 1.30 $ $Date: 1994/03/16 01:59:59 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"


static void OpenPipeline (pfPipe *p);


/*
 *   Usage() -- print usage advice and exit. This
 *   procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: simple file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float        t = 0.0f;
    pfScene     *scene;
    pfPipe      *p;
    pfChannel   *chan;
    pfNode      *root;

    if (argc < 2)
        Usage();

    /* Initialize Performer */
```

```
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH files in /usr/src/Performer/data */
pfFilePath(".:/usr/src/Performer/data");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
    pfExit();
    exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
chan = pfNewChan(p);
pfChanScene(chan, scene);
pfChanNearFar(chan, 1.0f, 1000.0f);
pfChanFOV(chan, 45.0f, 0.0f);

pfInitClock (0.0f);

/* Simulate for twenty seconds. */
while (t < 20.0f)
{
    float      s, c;
    pfCoord      view;

    /* Go to sleep until next frame time. */
    pfSync();

    /* Compute new view position. */
```

```
        t = pfGetTime();
        pfSinCos(45.0f*t, &s, &c);
        pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
        pfSetVec3(view.xyz, 100.0f*s, -110.0f*c, 30.0f);
        pfChanView(chan, view.xyz, view.hpr);

        /* Initiate cull/draw for this frame. */
        pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *   OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

2. \PTG\examples\OpenGVS\simple\simple.c

```
/*
 * simple.c: simple Performer program for programmer's guide
 *
 * $Revision: 1.30 $ $Date: 1994/03/16 01:59:59 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>



/* Any platform dependent includes appear here.
 * This should be the _only_ place to associate
 * platforms in the code. Review makefiles.
 */

#include "../../../Current/include/pfToGVS.h"

/*
#include <gl/device.h>
#include <Performer/pf.h>
#include "pfsgi.h"
*/

/* End all platform dependent stuff. */


 /*static*/ void OpenPipeline (pfPipe *p);


/*
 *      Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: simple file.ext ...\n");
    exit(1);
}


int
/* can only have one "main" - pfToGVS
```

```
 * needs it. Comment out your main()
 * and call it pfMain()
 *
main */pfMain( int argc, char *argv[] )

{
/*    float       t = 0.0f;       */
      pfScene     *scene;
      pfPipe      *p;
/*    pfChannel   *chan;          */
      pfNode      *root;

    if (argc < 2)
        Usage();

    /* Initialize Performer */
    pfInit();

    /* Use default multiprocessing mode based on number of
     * processors.
     */
    pfMultiprocess(PFMP_DEFAULT);

    /* Configure multiprocessing mode and start parallel
     * processes.
     */
    pfConfig();

    /* Append to PFPATH files in /usr/src/Performer/data */
    pfFilePath(".:/usr/src/Performer/data:$GV_ROOT/gvm");
    /* Read a single file, of any known type. */
    if ((root = LoadFile(argv[1], NULL)) == NULL)
    {
      pfExit();
      exit(-1);
    }

    /* Attach loaded file to a pfScene. */
    scene = pfNewScene();
    pfAddChild(scene, root);

    /* Create a pfLightSource and attach it to scene. */
    pfAddChild(scene, pfNewLSource());

    /* Configure and open GL window */
    p = pfGetPipe(0);
    pfInitPipe(p, OpenPipeline);

    /* Create and configure a pfChannel. */
    si_chan = pfNewChan(p);
    pfChanScene(si_chan, scene);
```

```
    pfChanNearFar(si_chan, 1.0f, 1000.0f);
    pfChanFOV(si_chan, 45.0f, 0.0f);

    pfInitClock (0.0f);

    /* here is pf loop */
    return G_SUCCESS;

/* Simulate for twenty seconds. */
/*  while (t < 20.0f)
    {
      float      s, c;
      pfCoord    view;

      Go to sleep until next frame time.
      pfSync();

      Compute new view position.
      t = pfGetTime();
      pfSinCos(45.0f*t, &s, &c);
      pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
      pfSetVec3(view.xyz, 100.0f*s, -110.0f*c, 30.0f);
      pfChanView(si_chan, view.xyz, view.hpr);

      Initiate cull/draw for this frame.
      pfFrame();
    }
 */
    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}

/*
 *      OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
/*static*/ void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);
```

```
    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

a. \PTG\examples\OpenGVS\simple\ptgproj.c

```
/*****************************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************************/


#include "../../../Current/include/pfToGVS.h"

/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix. See PTG Users Manual.
 * f - 5feb96
 */
int GV_user_proc( void )
{


/*
**
** Put simulation loop in here.
**
**
*/
     float      s, c;
     pfCoord    view;

    /*Go to sleep until next frame time.*/
     pfSync();

     /*Compute new view position.*/
     si_t = pfGetTime();
     pfSinCos(45.0f*si_t, &s, &c);
     pfSetVec3(view.hpr, 45.0f*si_t, -10.0f, 0);
     pfSetVec3(view.xyz, 100.0f*s, -110.0f*c, 30.0f);
     pfChanView(si_chan, view.xyz, view.hpr);

     /*Initiate cull/draw for this frame.*/
     pfFrame();

    return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\simple\ptgproj.h

```
/*****************************************************************************
*
*  ptgproj.h --
*
*    NPS Performer to OpenGVS Project globals for the user_proc loop
go here
*
*****************************************************************************/

#ifndef __PTGPROJ_H
#define __PTGPROJ_H


 __Globalutils__  float       si_t;
 __Globalutils__  pfChannel   *si_chan;

#endif
```

B. INHERIT DEMO

1. \PTG\examples\Performer\inherit.c

```
/*
 * inherit.c: Performer program to demonstrate use of inherit.
 *      Based on simple.c
 *
 * $Revision: 1.7 $ $Date: 1994/03/16 01:54:46 $
 *
 */

#include <math.h>
#include <stdlib.h>
#include <Performer/pf.h>
#include <Performer/pr.h>
#include "pfsgi.h"

char    file_path[256] = "/usr/src/Performer/data";

int
main(int argc, char *argv[])
{
    pfPipe          *Pipe0;
    pfScene         *Scene;
    pfChannel       *chan0;
    void            *arena;
    pfCoord          view;
    float            z, s, c;
    pfGroup         *group;
    pfGeoState      *gst1, *gst2;
    pfNode          *node1, *node2;
    pfDCS           *dcs1, *dcs2, *dcs3, *dcs4;
    pfMaterial      *mt1, *mt2;

    pfInit();             /* Initialize Performer */
    pfFilePath(file_path);
    pfMultiprocess(PFMP_APPCULLDRAW);/* Single thread for simplicity */
    pfConfig();             /* Configure */

    Pipe0 = pfGetPipe(0);
    chan0 = pfNewChan(Pipe0);

    arena = pfGetSharedArena();

    /* Create 1st geostate */
    gst1 = pfNewGState(arena);
    mt1 = pfNewMtl(arena);
    pfMtlColor(mt1, PFMTL_DIFFUSE, 0.8f, 0.8f, 0.0f);
```

```
pfMtlAlpha(mt1, 0.4f);
pfGStateAttr(gst1, PFSTATE_FRONTMTL, mt1);

/* Create 2nd geostate */
gst2 = pfNewGState(arena);
mt2 = pfNewMtl(arena);
pfMtlColor(mt2, PFMTL_DIFFUSE, 0.7f, 0.0f, 1.0f);
pfMtlAlpha(mt2, 0.8f);
pfGStateAttr(gst2, PFSTATE_FRONTMTL, mt2);
pfGStateMode(gst2, PFSTATE_TRANSPARENCY, PFTR_ON);

/* Load the files */
if ((node1 = LoadFile("f-117.dxf", gst1)) == NULL)
{
    pfExit();
    exit(-1);
}

if ((node2 = LoadFile("cow.obj", gst2)) == NULL)
{
    pfExit();
    exit(-1);
}

/* Create the nodes */
Scene = pfNewScene();
group = pfNewGroup();
dcs1 = pfNewDCS();
dcs2 = pfNewDCS();
dcs3 = pfNewDCS();
dcs4 = pfNewDCS();

/* Create the hierarchy */
pfAddChild(Scene, group);

pfAddChild(dcs1, node2);
pfAddChild(Scene, dcs1);

pfAddChild(dcs2, node2);
pfAddChild(dcs1, dcs2);
pfDCSScale(dcs2,.5f);

pfAddChild(dcs3, node1);
pfAddChild(group, dcs3);

pfAddChild(dcs4, node1);
pfAddChild(dcs3, dcs4);
pfDCSScale(dcs4,.5f);

pfInitPipe(Pipe0, NULL);
pfChanScene(chan0, Scene);
```

```
    /* Turn on lighting */
    pfEnable(PFEN_LIGHTING);
    pfApplyLModel(pfNewLModel(arena));
    pfLightOn(pfNewLight(arena));

    pfSetVec3(view.xyz, 0.0f, 0.0f, 50.0f);
    pfSetVec3(view.hpr, 0.0f, -90.0f, 0.0f);
    pfChanView(chan0, view.xyz, view.hpr);

    /* Loop through various transformations of the DCS's */
    for (z = 0.; z < 1084; z += 4.)
    {
        pfDCSRot(dcs1,
            (z < 360) ? (int) z % 360 : 0.,
            (z > 360 && z < 720) ? (int) z % 360 : 0.,
            (z > 720) ? (int) z % 360 : 0.);

        pfSinCos(z, &s, &c);
        pfDCSTrans(dcs2, 5.0f * c, 5.0f * s, 0.f);

        pfDCSRot(dcs3, z, 0, 0);
        pfDCSTrans(dcs3, 15.0f * c, 15.0f * s, 0.f * s);
        pfDCSRot(dcs4, 0, 0, z);
        pfDCSTrans(dcs4, 4.0f * c, 4.0f * s, 0.f);

        pfFrame();
    }
    sleep(3);
    pfExit();
    exit(0);
}
```

2. \PTG\examples\OpenGVS\inherit\inherit.c

```
/*
#include <math.h>
#include <stdlib.h>
#include <Performer/pf.h>
#include <Performer/pr.h>
#include "pfsgi.h"
*/

#include "../../../Current/include/pfToGVS.h"

char  file_path[256] = ":$GV_ROOT/gvm";

int
/*
main
*/
pfMain(int argc, char *argv[])
{
    pfPipe          *Pipe0;
    pfScene         *Scene;
    pfChannel       *chan0;
    void            *arena;
    pfCoord          view;
/*  float            z, s, c;     */
    pfGroup         *group;
    pfGeoState      *gst1, *gst2;
    pfNode          *node1, *node2;
/*  pfDCS           *dcs1, *dcs2, *dcs3, *dcs4;    */
    pfMaterial      *mt1, *mt2;

    pfInit();     /* Initialize Performer */
    pfFilePath(file_path);
    pfMultiprocess(PFMP_APPCULLDRAW); /* Single thread for simplicity */
    pfConfig();     /* Configure */

    Pipe0 = pfGetPipe(0);
    chan0 = pfNewChan(Pipe0);

    arena = pfGetSharedArena();

    /* Create 1st geostate */
    gst1 = pfNewGState(arena);
    mt1 = pfNewMtl(arena);
    pfMtlColor(mt1, PFMTL_DIFFUSE, 0.8f, 0.8f, 0.0f);
    pfMtlAlpha(mt1, 0.4f);
    pfGStateAttr(gst1, PFSTATE_FRONTMTL, mt1);
```

```
/* Create 2nd geostate */
gst2 = pfNewGState(arena);
mt2 = pfNewMtl(arena);
pfMtlColor(mt2, PFMTL_DIFFUSE, 0.7f, 0.0f, 1.0f);
pfMtlAlpha(mt2, 0.8f);
pfGStateAttr(gst2, PFSTATE_FRONTMTL, mt2);
pfGStateMode(gst2, PFSTATE_TRANSPARENCY, PFTR_ON);

/* Load the files */
if ((node1 = LoadFile("yf23.gvm", NULL)) == NULL)
{
   pfExit();
   exit(-1);
}

if ((node2 = LoadFile("truck.gvm", gst2)) == NULL)
{
   pfExit();
   exit(-1);
}

/* Create the nodes */
Scene = pfNewScene();
group = pfNewGroup();
ih_dcs1 = pfNewDCS();
ih_dcs2 = pfNewDCS();
ih_dcs3 = pfNewDCS();
ih_dcs4 = pfNewDCS();

/* Create the hierarchy */
pfAddChild(Scene, group);

pfAddChild(ih_dcs1, node2);
pfAddChild(Scene, ih_dcs1);

pfAddChild(ih_dcs2, node2);
pfAddChild(ih_dcs1, ih_dcs2);
pfDCSScale(ih_dcs2,.5f);

pfAddChild(ih_dcs3, node1);
pfAddChild(group, ih_dcs3);

pfAddChild(ih_dcs4, node1);
pfAddChild(ih_dcs3, ih_dcs4);
pfDCSScale(ih_dcs4,.5f);

pfInitPipe(Pipe0, NULL);
pfChanScene(chan0, Scene);

/* Turn on lighting */
pfEnable(PFEN_LIGHTING);
```

```
    pfApplyLModel(pfNewLModel(arena));
    pfLightOn(pfNewLight(arena));

    pfSetVec3(view.xyz, 0.0f, 0.0f, 50.0f);
    pfSetVec3(view.hpr, 0.0f, -90.0f, 0.0f);
    pfChanView(chan0, view.xyz, view.hpr);

    return G_SUCCESS;

 /* Loop through various transformations of the DCS's */
 /* for ( z = 0.; z < 1084; z += 4.)
    {
       pfDCSRot(dcs1,
         (z < 360) ? (int) z % 360 : 0.,
         (z > 360 && z < 720) ? (int) z % 360 : 0.,
         (z > 720) ? (int) z % 360 : 0.);

         pfSinCos(z, &s, &c);
         pfDCSTrans(dcs2, 5.0f * c, 5.0f * s, 0.f);

         pfDCSRot(dcs3, z, 0, 0);
         pfDCSTrans(dcs3, 15.0f * c, 15.0f * s, 0.f * s);
         pfDCSRot(dcs4, 0, 0, z);
         pfDCSTrans(dcs4, 4.0f * c, 4.0f * s, 0.f);

         pfFrame();
    }
    sleep(3);
    pfExit();
    exit(0);
*/
}
```

a. \PTG\examples\OpenGVS\inherit\ptgproj.c

```
/*****************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************/

#include "../../../Current/include/pfToGVS.h"

/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix.
 * f - 5feb96
 */
int GV_user_proc( void )
{

/*
**
** Put simulation loop in here.
**
**
*/
   static float z = -4.0;

   z += 4.0;

   pfDCSRot(ih_dcs1,
      (z < 360) ? (int) z % 360 : 0.,
      (z > 360 && z < 720) ? (int) z % 360 : 0.,
      (z > 720) ? (int) z % 360 : 0.);

   pfSinCos(z, &ih_s, &ih_c);
   pfDCSTrans(ih_dcs2, 5.0f * ih_c, 5.0f * ih_s, 0.f);

   pfDCSRot(ih_dcs3, z, 0, 0);
   pfDCSTrans(ih_dcs3, 15.0f * ih_c, 15.0f * ih_s, 0.f * ih_s);
   pfDCSRot(ih_dcs4, 0, 0, z);
   pfDCSTrans(ih_dcs4, 4.0f * ih_c, 4.0f * ih_s, 0.f);

   pfFrame();

   return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\inherit\ptgproj.h

```
/*****************************************************************
*
*  ptgproj.h --
*
*       NPS Performer to OpenGVS Project Simulation Loop Header File
*
*****************************************************************/

/* ptgproj.h - > globals for the user_proc loop go here */

#ifndef __PTGPROJ_H
#define __PTGPROJ_H


__Globalutils__ pfDCS          *ih_dcs1, *ih_dcs2, *ih_dcs3, *ih_dcs4;
__Globalutils__ float           ih_z, ih_s, ih_c;

#endif
```

C. MULTIPLE CHANNEL DEMO

1. \PTG\examples\Performer\multichan.c

```
/*
 * multichan.c: Performer program to demonstrate multiple channels
 *              in one pipe.  Derived from simple.c
 *
 * $Revision: 1.5 $ $Date: 1994/03/16 01:59:34 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"


static void OpenPipeline (pfPipe *p);


/*
 *   Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: multichan file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float        t = 0.0f;
    pfScene     *scene;
    pfPipe      *p;
    pfChannel   *chan[4];
    pfNode      *root;
    pfSphere     bsphere;
    int          loop;

    if (argc < 2)
```

```
    Usage();

/* Initialize Performer */
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath(".:/usr/src/Performer/data");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
    pfExit();
    exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &bsphere);
printf("bsphere.radius = %f\n",bsphere.radius);
/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
for (loop=0; loop < 4; loop++)
{
    chan[loop] = pfNewChan(p);
    pfChanScene(chan[loop], scene);
    pfChanNearFar(chan[loop], 1.0f, 10.0f * bsphere.radius);
    pfChanFOV(chan[loop], 45.0f, 0.0f);
}

pfChanViewport (chan[0], 0.0, 0.5, 0.0, 0.5);
```

```
    pfChanViewport (chan[1], 0.5, 1.0, 0.0, 0.5);
    pfChanViewport (chan[2], 0.5, 1.0, 0.5, 1.0);
    pfChanViewport (chan[3], 0.0, 0.5, 0.5, 1.0);

    pfInitClock (0.0f);

    /* Simulate for twenty seconds. */
    while (t < 20.0f)
    {
        float      s, c;
        pfCoord      view;

        /* Go to sleep until next frame time. */
        pfSync();

        /* Compute new view position. */
        t = pfGetTime();
        pfSinCos(45.0f*t, &s, &c);
        pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
        pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
            -2.0f * bsphere.radius *c,
            0.5f * bsphere.radius);

        for (loop=0; loop < 4; loop++)
            pfChanView(chan[loop], view.xyz, view.hpr);

        /* Initiate cull/draw for this frame. */
        pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *   OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");
```

```
    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

2. \PTG\examples\OpenGVS\multichan\multichan.c

```
/*
 * multichan.c: Performer program to demonstrate multiple channels
 *              in one pipe.  Derived from simple.c
 *
 * $Revision: 1.5 $ $Date: 1994/03/16 01:59:34 $
 *
 */

/*
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"
*/

#include "../../../Current/include/pfToGVS.h"

static void OpenPipeline (pfPipe *p);


/*
 * Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: multichan file.ext ...\n");
    exit(1);
}


int
/*main*/ pfMain (int argc, char *argv[])
{
/*   float        t = 0.0f;   */
     pfScene      *scene;
     pfPipe       *p;
/*   pfChannel    *chan[4];   */
     pfNode       *root;
/*   pfSphere     bsphere;    */
/*   int          loop;       */

    if (argc < 2)
```

```
   Usage();

/* Initialize Performer */
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath( ".:/usr/src/Performer/data:$GV_ROOT/gvm");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
   pfExit();
   exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &mc_bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
for (mc_loop=0; mc_loop < 4; mc_loop++)
{
  mc_chan[mc_loop] = pfNewChan(p);
  pfChanScene(mc_chan[mc_loop], scene);
  pfChanNearFar(mc_chan[mc_loop], 1.0f, 10.0f * mc_bsphere.radius);
  pfChanFOV(mc_chan[mc_loop], 45.0f, 0.0f);
}

pfChanViewport (mc_chan[0], 0.0, 0.5, 0.0, 0.5);
```

```
    pfChanViewport (mc_chan[1], 0.5, 1.0, 0.0, 0.5);
    pfChanViewport (mc_chan[2], 0.5, 1.0, 0.5, 1.0);
    pfChanViewport (mc_chan[3], 0.0, 0.5, 0.5, 1.0);

    pfInitClock (0.0f);

    return G_SUCCESS;

/*   Simulate for twenty seconds. */
/*   while (t < 20.0f)
     {
         float      s, c;
         pfCoord    view;

         Go to sleep until next frame time.
         pfSync();

         Compute new view position.
         t = pfGetTime();
         pfSinCos(45.0f*t, &s, &c);
         pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
         pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
            -2.0f * bsphere.radius *c,
            0.5f * bsphere.radius);

         for (loop=0; loop < 4; loop++)
         pfChanView(chan[loop], view.xyz, view.hpr);

         Initiate cull/draw for this frame.
         pfFrame();
    }
*/
    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 * OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
```

```
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

a. \PTG\examples\OpenGVS\multichan\ptgproj.c

```
/*****************************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************************/

#include "../../../Current/include/pfToGVS.h"

/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix. See PTG Users Manual.
 * f - 5feb96
 */
int GV_user_proc( void )
{
/*
**
** Put simulation loop in here.
**
**
*/
   float      s, c;
   pfCoord    view;

   /*Go to sleep until next frame time. */
   pfSync();

   /*Compute new view position.      */
   mc_t = pfGetTime();
   pfSinCos(45.0f*mc_t, &s, &c);
   pfSetVec3(view.hpr, 45.0f*mc_t, -10.0f, 0);
   pfSetVec3(view.xyz, 2.0f * mc_bsphere.radius * s,
      -2.0f * mc_bsphere.radius *c,
      0.5f * mc_bsphere.radius);

   for (mc_loop=0; mc_loop < 4; mc_loop++)
   pfChanView(mc_chan[mc_loop], view.xyz, view.hpr);

   /*Initiate cull/draw for this frame. */
   pfFrame();

   return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\multichan\ptgproj.h

```
/*****************************************************************************
*
*  ptgproj.h --
*
*    NPS Performer to OpenGVS Project globals for the user_proc loop
go here
*
*****************************************************************************/

#ifndef __PTGPROJ_H
#define __PTGPROJ_H

 /*
__Globalutils__ type1 var1;
__Globalutils__ type2 var2;
 */

__Globalutils__ float       mc_t/* = 0.0f*/;
__Globalutils__ pfChannel   *mc_chan[4];
__Globalutils__ pfSphere    mc_bsphere;
__Globalutils__ int         mc_loop;

#endif
```

D. MULTIPLE PIPE DEMO

1. \PTG\examples\Performer\multipipe.c

```
/*
 * multipipe.c: simple Performer program to demonstrate use of
 *              multiple pfPipe's.  based on simple.c
 *
 * $Revision: 1.5 $ $Date: 1994/03/16 01:59:46 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"

static void OpenPipeline (pfPipe *p);

/*
 *   Usage() -- print usage advice and exit. This
 *   procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: multipipe file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float        t = 0.0f;
    pfScene     *scene;
    pfPipe      *pipe[4];
    pfChannel   *chan[4];
    pfNode      *root;
    pfSphere     bsphere;
    int          loop;

    if (argc < 2)
        Usage();
```

```
/* Initialize Performer */
pfInit();

/* specify the number of pfPipes */
pfMultipipe (4);

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath(".:/usr/src/Performer/data");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
    pfExit();
    exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL windows */
for (loop=0; loop < 4; loop++)
{
    pipe[loop] = pfGetPipe(loop);
    pfInitPipe(pipe[loop], OpenPipeline);
}

/* Create and configure pfChannels. */
for (loop=0; loop < 4; loop++)
{
    chan[loop] = pfNewChan(pipe[loop]);
    pfChanScene(chan[loop], scene);
    pfChanNearFar(chan[loop], 1.0f, 10.0f * bsphere.radius);
```

```
        pfChanFOV(chan[loop], 45.0f, 0.0f);
    }
    pfInitClock (0.0f);

    /* Simulate for twenty seconds. */
    while (t < 20.0f)
    {
        float       s, c;
        pfCoord     view;

        /* Go to sleep until next frame time. */
        pfSync();

        /* Compute new view position. */
        t = pfGetTime();
        pfSinCos(45.0f*t, &s, &c);
        pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
        pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
            -2.0f * bsphere.radius *c,
            0.5f * bsphere.radius);

        for (loop=0; loop < 4; loop++)
            pfChanView(chan[loop], view.xyz, view.hpr);

        /* Initiate cull/draw for this frame. */
        pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *   OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();

    /* prefposition(100, 500, 100, 500); */

    winopen("IRIS Performer");
```

```
    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

2. \PTG\examples\OpenGVS\multipipe\multipipe.c

```
/*
 * multipipe.c: simple Performer program to demonstrate use of
 *              multiple pfPipe's.  based on simple.c
 *
 * $Revision: 1.5 $ $Date: 1994/03/16 01:59:46 $
 *
 */

/*
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"
*/

#include "../../../Current/include/pfToGVS.h"

static void OpenPipeline (pfPipe *p);


/*
 *     Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: multipipe file.ext ...\n");
    exit(1);
}


int
pfMain (int argc, char *argv[])
{
/*    float       t = 0.0f;     */
      pfScene     *scene;
      pfPipe      *pipe[4];
/*    pfChannel   *chan[4];     */
      pfNode      *root;
/*    pfSphere    bsphere;      */
/*    int         loop;         */
```

```
if (argc < 2)
  Usage();

/* Initialize Performer */
pfInit();

/* specify the number of pfPipes */
pfMultipipe (4);

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath(".:/usr/src/Performer/data:$GV_ROOT/gvm");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
     pfExit();
     exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &mc_bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL windows */
for (mc_loop=0; mc_loop < 4; mc_loop++)
{
     pipe[mc_loop] = pfGetPipe(mc_loop);
     pfInitPipe(pipe[mc_loop], OpenPipeline);
}

/* Create and configure pfChannels. */
for (mc_loop=0; mc_loop < 4; mc_loop++)
{
```

```
        mc_chan[mc_loop] = pfNewChan(pipe[mc_loop]);
        pfChanScene(mc_chan[mc_loop], scene);
        pfChanNearFar(mc_chan[mc_loop], 1.0f, 10.0f *
mc_bsphere.radius);
        pfChanFOV(mc_chan[mc_loop], 45.0f, 0.0f);
    }

    pfInitClock (0.0f);

    return G_SUCCESS;
/*
   Simulate for twenty seconds.
   while (t < 20.0f)
   {
      float          s, c;
      pfCoord        view;

      Go to sleep until next frame time.
      pfSync();

      Compute new view position.
      t = pfGetTime();
      pfSinCos(45.0f*t, &s, &c);
      pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
      pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
         -2.0f * bsphere.radius *c,
         0.5f * bsphere.radius);

      for (loop=0; loop < 4; loop++)
         pfChanView(chan[loop], view.xyz, view.hpr);

      Initiate cull/draw for this frame.
      pfFrame();
   }
*/
   /* Terminate parallel processes and exit. */
   pfExit();

   return 0;
}


/*
 *     OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
```

```
    /* Open graphics window. */
    foreground();

    /* prefposition(100, 500, 100, 500); */

    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

a. *\PTG\examples\OpenGVS\multipipe\ptgproj.c*

```
/*****************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************/

#include "../../../Current/include/pfToGVS.h"

/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix. See PTG Users Manual.
 * f - 5feb96
 */
int GV_user_proc( void )
{
/*
**
** Put simulation loop in here.
**
**
*/
   float      s, c;
   pfCoord    view;

   /*Go to sleep until next frame time. */
   pfSync();

   /*Compute new view position.      */
   mc_t = pfGetTime();
   pfSinCos(45.0f*mc_t, &s, &c);
   pfSetVec3(view.hpr, 45.0f*mc_t, -10.0f, 0);
   pfSetVec3(view.xyz, 2.0f * mc_bsphere.radius * s,
      -2.0f * mc_bsphere.radius *c,
      0.5f * mc_bsphere.radius);

   for (mc_loop=0; mc_loop < 4; mc_loop++)
      pfChanView(mc_chan[mc_loop], view.xyz, view.hpr);

   /*Initiate cull/draw for this frame. */
   pfFrame();

    return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\multipipe\ptgproj.h

```
/*****************************************************************************
*
*  ptgproj.h --
*
*    NPS Performer to OpenGVS Project globals for the user_proc loop
go here
*
*****************************************************************************/

#ifndef __PTGPROJ_H
#define __PTGPROJ_H

 /*
__Globalutils__ type1 var1;
__Globalutils__ type2 var2;
 */
__Globalutils__ float       mc_t /* = 0.0f*/;
__Globalutils__ pfChannel   *mc_chan[4];
__Globalutils__ pfSphere    mc_bsphere;
__Globalutils__ int         mc_loop;

#endif
```

E. EARTHSKY DEMO

1. \PTG\examples\Performer\earthsky.c

```
/*
 * earthsky.c: Performer program to demonstrate use of earthsky.
 *      Based on simple.c
 *
 * $Revision: 1.7 $ $Date: 1994/03/16 01:54:46 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"


static void OpenPipeline (pfPipe *p);
static void DrawChannel (pfChannel *chan, void *data);

/*
 *   Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    printf(stderr, "Usage: earthsky file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float       t = 0.0f;
    pfScene     *scene;
    pfPipe      *p;
    pfChannel   *chan;
    pfNode      *root;
    pfSphere    bsphere;
    pfEarthSky  *esky;

    if (argc < 2)
```

```
    Usage();

/* Initialize Performer */
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath(".:/usr/src/Performer/data");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
    pfExit();
    exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
chan = pfNewChan(p);
pfChanScene(chan, scene);
pfChanNearFar(chan, 1.0f, 10.0f * bsphere.radius);
pfChanFOV(chan, 45.0f, 0.0f);
pfChanDrawFunc(chan, DrawChannel);

esky = pfNewESky();
pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
pfESkyAttr(esky, PFES_GRND_HT, -1.0f * bsphere.radius);
```

```
    pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
    pfESkyColor(esky, PFES_GRND_NEAR, 0.5f, 0.3f, 0.1f, 1.0f);
    pfChanESky(chan, esky);

    pfInitClock(0.0f);

    /* Simulate for twenty seconds. */
    while (t < 20.0f)
    {
        float      s, c;
        pfCoord      view;

        /* Go to sleep until next frame time. */
        pfSync();

        /* Compute new view position. */
        t = pfGetTime();
        pfSinCos(45.0f*t, &s, &c);
        pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
        pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
            -2.0f * bsphere.radius *c,
            0.5f * bsphere.radius);
        pfChanView(chan, view.xyz, view.hpr);

        /* Initiate cull/draw for this frame. */
        pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *   OpenPipeline() -- create a GL window: set up the
 *   window system, IRIS GL, and IRIS Performer. This
 *   procedure is executed in the draw process (when
 *   there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);
```

```
    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}

/* Draw process callback */
static void DrawChannel (pfChannel *chan, void *data)
{
    pfClearChan (chan);
    pfDraw ();
}
```

2. \PTG\examples\OpenGVS\earthsky\earthsky.c

```
/*
 * earthsky.c: Performer program to demonstrate use of earthsky.
 *      Based on simple.c
 *
 * $Revision: 1.7 $ $Date: 1994/03/16 01:54:46 $
 *
 */
#include "../../../Current/include/pfToGVS.h"
/*
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"
*/

static void OpenPipeline (pfPipe *p);
static void DrawChannel (pfChannel *chan, void *data);

/*
 * Usage() -- print usage advice and exit. This
 * procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: earthsky file.ext ...\n");
    exit(1);
}


int
pfMain (int argc, char *argv[])
{
 /*   float          t = 0.0f;*/
      pfScene        *scene;
      pfPipe         *p;
 /*   pfChannel      *chan;   */
      pfNode         *root;
 /*   pfSphere       bsphere; */
      pfEarthSky     *esky;

    if (argc < 2)
       Usage();
```

```
/* Initialize Performer */
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath(".:/usr/src/Performer/data:$GV_ROOT/gvm");

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
   pfExit();
   exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, &es_bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
es_chan = pfNewChan(p);
pfChanScene(es_chan, scene);
pfChanNearFar(es_chan, 1.0f, 10.0f * es_bsphere.radius);
pfChanFOV(es_chan, 45.0f, 0.0f);
pfChanDrawFunc(es_chan, DrawChannel);

esky = pfNewESky();
pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
pfESkyAttr(esky, PFES_GRND_HT, -1.0f * es_bsphere.radius);
pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
```

```
    pfESkyColor(esky, PFES_GRND_NEAR, 0.5f, 0.3f, 0.1f, 1.0f);
    pfChanESky(es_chan, esky);

    pfInitClock(0.0f);

return G_SUCCESS;

    /* Simulate for twenty seconds.
    while (t < 20.0f)
    {
       float      s, c;
       pfCoord    view;

       Go to sleep until next frame time.
       pfSync();

       Compute new view position.
       t = pfGetTime();
       pfSinCos(45.0f*t, &s, &c);
       pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
       pfSetVec3(view.xyz, 2.0f * bsphere.radius * s,
         -2.0f * bsphere.radius *c,
         0.5f * bsphere.radius);
       pfChanView(chan, view.xyz, view.hpr);

      Initiate cull/draw for this frame.
      pfFrame();
    }

    Terminate parallel processes and exit.
    pfExit();

    return 0;*/
}


/*
 * OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
Open Pipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults.*/
```

```
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model.*/
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}

/* Draw process callback */
static void DrawChannel (pfChannel *chan, void *data)
{
   pfClearChan (chan);
   pfDraw ();
}
```

a. \PTG\examples\OpenGVS\earthsky\ptgproj.c

```
/*****************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************/

#include "../../../Current/include/pfToGVS.h"
/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix.
 * f - 5feb96
 */
int GV_user_proc( void )
{


/*
**
** Put simulation loop in here.
**
**
*/
   float      s, c;
   pfCoord    view;

   /* Go to sleep until next frame time. */
   pfSync();

   /* Compute new view position. */
   es_t = pfGetTime();
   pfSinCos(45.0f*es_t, &s, &c);
   pfSetVec3(view.hpr, 45.0f*es_t, -10.0f, 0);
   pfSetVec3(view.xyz, 2.0f * es_bsphere.radius * s,
      -2.0f * es_bsphere.radius *c,
      0.5f * es_bsphere.radius);
   pfChanView(es_chan, view.xyz, view.hpr);

   /* Initiate cull/draw for this frame. */
   pfFrame();

   return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\earthsky\ptgproj.h

```
/*****************************************************************
*
*  ptgproj.h --
*
*    NPS Performer to OpenGVS Project globals for the user_proc loop
go here
*
*****************************************************************/


/* ptgproj.h - > globals for the user_proc loop go here */

#ifndef __PTGPROJ_H
#define __PTGPROJ_H

__Globalutils__ float         es_t;
__Globalutils__  pfSphere     es_bsphere;
__Globalutils__  pfChannel    *es_chan;

#endif
```

F. FOG DEMO

1. \PTG\examples\Performer\fog.c

```
/*
 * fog.c: Performer program to demonstrate use of fog.
 *        Based on simple.c
 *
 * $Revision: 1.7 $ $Date: 1994/03/16 01:54:56 $
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"


static void OpenPipeline (pfPipe *p);
static void DrawChannel (pfChannel *chan, void *data);

pfSphere *bsphere;

/*
 *   Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: fog file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float         t = 0.0f;
    pfScene       *scene;
    pfPipe        *p;
    pfChannel     *chan;
    pfEarthSky    *esky;
    pfNode        *root;
```

```
    if (argc < 2)
        Usage();

    /* Initialize Performer */
    pfInit();

    /* Use default multiprocessing mode based on number of
     * processors.
     */
    pfMultiprocess(PFMP_DEFAULT);
    bsphere = (pfSphere*) pfMalloc (sizeof(pfSphere),
pfGetSharedArena());

    /* Configure multiprocessing mode and start parallel
     * processes.
     */
    pfConfig();

    /* Append to PFPATH additional standard directories where
     * geometry and textures exist
     */
    pfFilePath(".:/usr/src/Performer/data");

    /* Read a single file, of any known type. */
    if ((root = LoadFile(argv[1], NULL)) == NULL)
    {
        pfExit();
        exit(-1);
    }

    /* Attach loaded file to a pfScene. */
    scene = pfNewScene();
    pfAddChild(scene, root);

    /* determine extent of scene's geometry */
    pfGetNodeBSphere (scene, bsphere);

    /* Create a pfLightSource and attach it to scene. */
    pfAddChild(scene, pfNewLSource());

    /* Configure and open GL window */
    p = pfGetPipe(0);
    pfInitPipe(p, OpenPipeline);

    /* Create and configure a pfChannel. */
    chan = pfNewChan(p);
    pfChanScene(chan, scene);
    pfChanNearFar(chan, 1.0f, 10.0f * bsphere->radius);
    pfChanFOV(chan, 45.0f, 0.0f);
    pfChanDrawFunc(chan, DrawChannel);
```

```
    esky = pfNewESky();
    pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
    pfESkyAttr(esky, PFES_GRND_HT, -1.0f * bsphere->radius);
    pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
    pfESkyColor(esky, PFES_GRND_NEAR, 0.5f, 0.3f, 0.1f, 1.0f);
    pfChanESky(chan, esky);

    pfInitClock (0.0f);

    /* Simulate for twenty seconds. */
    while (t < 20.0f)
    {
        float      s, c;
        pfCoord      view;

        /* Go to sleep until next frame time. */
        pfSync();

        /* Compute new view position. */
        t = pfGetTime();
        pfSinCos(45.0f*t, &s, &c);
        pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
        pfSetVec3(view.xyz, 2.0f * bsphere->radius * s,
            -2.0f * bsphere->radius *c,
            0.5f * bsphere->radius);

        pfChanView(chan, view.xyz, view.hpr);

        /* Initiate cull/draw for this frame. */
        pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *   OpenPipeline() -- create a GL window: set up the
 *    window system, IRIS GL, and IRIS Performer. This
 *    procedure is executed in the draw process (when
 *    there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    pfFog *fog;

    /* Open graphics window. */
```

```
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));

    fog = pfNewFog(pfGetSharedArena());
    pfFogType (fog, PFFOG_VTX_LIN);
    pfFogColor (fog, 0.0, 0.0, 0.0);
    pfFogRange (fog, 1.0f * bsphere->radius, 4.0f * bsphere->radius);
    pfApplyFog (fog);
    pfEnable (PFEN_FOG);
    pfOverride(PFSTATE_FOG | PFSTATE_ENFOG, PF_ON);
}

/* Draw process callback */
static void DrawChannel (pfChannel *chan, void *data)
{
    pfClearChan (chan);
    pfDraw ();
}
```

2. \PTG\examples\OpenGVS\fog\fog.c

```
/*
 * fog.c: Performer program to demonstrate use of fog.
 *        Based on simple.c
 *
 * $Revision: 1.7 $ $Date: 1994/03/16 01:54:56 $
 *
 */

#include "../../../Current/include/pfToGVS.h"

/*
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfsgi.h"

*/
static void OpenPipeline (pfPipe *p);
static void DrawChannel (pfChannel *chan, void *data);

/*
 * Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: fog file.ext ...\n");
    exit(1);
}


int
pfMain (int argc, char *argv[])
{

    pfScene       *scene;
    pfPipe        *p;
    pfEarthSky    *esky;
    pfNode        *root;

    if (argc < 2)
       Usage();
```

```
    /* Initialize Performer */
    pfInit();

    /* Use default multiprocessing mode based on number of
     * processors.
     */
    pfMultiprocess(PFMP_DEFAULT);
    bsphere = (pfSphere*) pfMalloc (sizeof(pfSphere),
pfGetSharedArena());

    /* Configure multiprocessing mode and start parallel
     * processes.
     */
    pfConfig();

    /* Append to PFPATH additional standard directories where
     * geometry and textures exist
     */
    pfFilePath(".:/usr/src/Performer/data"
               ":$GV_ROOT/gvm");

    /* Read a single file, of any known type. */

    if ((root = LoadFile(argv[1], NULL)) == NULL)
    {
       pfExit();
       exit(-1);
    }

    /* Attach loaded file to a pfScene. */
    scene = pfNewScene();
    pfAddChild(scene, root);

    /* determine extent of scene's geometry */
    pfGetNodeBSphere (scene, bsphere);

    /* Create a pfLightSource and attach it to scene. */
    pfAddChild(scene, pfNewLSource());

    /* Configure and open GL window */
    p = pfGetPipe(0);
    pfInitPipe(p, OpenPipeline);


    /* Create and configure a pfChannel. */
    chan = pfNewChan(p);
    pfChanScene(chan, scene);
    pfChanNearFar(chan, 1.0f, 10.0f * bsphere->radius);
    pfChanFOV(chan, 45.0f, 0.0f);
```

```
    pfChanDrawFunc(chan, DrawChannel);

    esky = pfNewESky();
    pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
    pfESkyAttr(esky, PFES_GRND_HT, -1.0f * bsphere->radius);
    pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
    pfESkyColor(esky, PFES_GRND_NEAR, 0.5f, 0.3f, 0.1f, 1.0f);
    pfChanESky(chan, esky);
    pfInitClock (0.0f);

    return G_SUCCESS;

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}

/*
 * OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    pfFog *fog;

    /* Open graphics window.*/
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults.*/
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));


    fog = pfNewFog(pfGetSharedArena());
    pfFogType (fog, PFFOG_VTX_LIN);
    pfFogColor (fog, 0.0, 0.0, 0.0);
    pfFogRange (fog, 1.0f * bsphere->radius, 4.0f * bsphere->radius);
    pfApplyFog (fog);
```

```
    pfEnable (PFEN_FOG);
    pfOverride(PFSTATE_FOG | PFSTATE_ENFOG, PF_ON);
}

/* Draw process callback */
static void DrawChannel (pfChannel *chan, void *data)
{
    pfClearChan (chan);
    pfDraw ();
}
```

a. \PTG\examples\OpenGVS\fog\ptgproj.c

```
/*****************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************/

#include "../../../Current/include/pfToGVS.h"
/*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix.
 * f - 5feb96
 */
int GV_user_proc( void )
{

/*
**
** Put simulation loop in here.
**
**
*/
    static float      s, c;
    static pfCoord    view;

    /* Go to sleep until next frame time. */
    pfSync();

    /* Compute new view position. */
    t = pfGetTime();
    pfSinCos(45.0f*t, &s, &c);
    pfSetVec3(view.hpr, 45.0f*t, -10.0f, 0);
    /*pfSetVec3(view.xyz, 100.0f*s, -110.0f*c, 30.0f);*/

    pfSetVec3(view.xyz, 2.0f * bsphere->radius * s,
     -2.0f * bsphere->radius *c, 0.5f * bsphere->radius);

    pfChanView(chan, view.xyz, view.hpr);

    /* Initiate cull/draw for this frame.*/
    pfFrame();

    return G_SUCCESS;
}
```

b. \PTG\examples\OpenGVS\fog\ptgproj.h

```
/*****************************************************************
*
*  ptgproj.h --
*
*    NPS Performer to OpenGVS Project globals for the user_proc loop
go here
*
*****************************************************************/

/* ptgproj.h - > globals for the user_proc loop go here */

#ifndef __PTGPROJ_H
#define __PTGPROJ_H

__Globalutils__ float       t;
__Globalutils__  pfSphere   *bsphere;
__Globalutils__  pfChannel  *chan;

#endif
```

G. SMOKE DEMO

1. \PTG\examples\Performer\smoke.c

```
/*
 * smoke.c:   simple Performer program to demonstrate using
 *            pfutil smoke
 *
 * $Revision: 1.6 $ $Date: 1994/03/16 09:35:33 $

 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <gl/device.h>

#include <Performer/pf.h>
#include "pfutil.h"
#include "pfsgi.h"


static void OpenPipeline (pfPipe *p);
static void DrawChannel(pfChannel *chan, void *data);

static pfMatrix idmat ={{1., 0., 0., 0.},
                        {0., 1., 0., 0.},
                        {0., 0., 1., 0.},
                        {0., 0., 0., 1.}};

pfSphere *bsphere;

static void DrawChannel (pfChannel *channel, void *data)
{
    pfVec3 vec;
        static pfMatrix tempmat;

        pfClearChan (channel);
        pfDraw();

    pfSetVec3 (vec, 0.0f, -1.0f, 0.0f);
    pfuDrawSmokes (vec);
}

/*
 *    Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
```

```
static void
Usage (void)
{
    fprintf(stderr, "Usage: smoke file.ext ...\n");
    exit(1);
}


int
main (int argc, char *argv[])
{
    float       t = 0.0f;
    pfScene     *scene;
    pfPipe      *p;
    pfChannel   *chan;
    pfNode    *root;
    pfEarthSky    *esky;
    pfVec3    vec;
    pfuSmoke    *smoke;
    pfuSmoke    *fire;

    if (argc < 2)
    Usage();

    /* Initialize Performer */
    pfInit();

    /* Use default multiprocessing mode based on number of
     * processors.
     */
    pfMultiprocess(PFMP_DEFAULT);
    bsphere = (pfSphere*) pfMalloc (sizeof(pfSphere), pfGetSharedAr-
ena());

    /* Append to PFPATH additional standard directories where
     * geometry and textures exist
     */
    pfFilePath(".:/usr/src/Performer/data");

    pfuInitSmokes();

    /* Configure multiprocessing mode and start parallel
     * processes.
     */
    pfConfig();

    /* Read a single file, of any known type. */
    if ((root = LoadFile(argv[1], NULL)) == NULL)
    {
    pfExit();
    exit(-1);
```

```
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
chan = pfNewChan(p);
pfChanDrawFunc(chan, DrawChannel);
pfChanScene(chan, scene);
pfChanNearFar(chan, 1.0f, 10.0f * bsphere->radius);
pfChanFOV(chan, 45.0f, 0.0f);

esky = pfNewESky();
pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
pfESkyAttr(esky, PFES_GRND_HT, -1.0f * bsphere->radius);
pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
pfESkyColor(esky, PFES_GRND_NEAR, 0.5f, 0.3f, 0.1f, 1.0f);
pfChanESky(chan, esky);

pfSetVec3 (vec, 0.0f, 0.0f, 0.0f);

fire = pfuNewSmoke();
pfuSmokeType (fire, PFUSMOKE_FIRE);
pfuSmokeOrigin (fire, vec, 0.25f * bsphere->radius);
pfuSmokeMode (fire, PFUSMOKE_START);

smoke = pfuNewSmoke();
pfuSmokeType (smoke, PFUSMOKE_SMOKE);
pfuSmokeOrigin (smoke, vec, 0.25f * bsphere->radius);
pfuSmokeVelocity (smoke, 1.0f, 0.25f * bsphere->radius);
pfuSmokeDensity (smoke, 0.33f, 2.42f, 1.83f);
pfuSmokeMode (smoke, PFUSMOKE_START);

/* Simulate for twenty seconds. */
while (t < 20.0f)
{
pfCoord    view;

/* Go to sleep until next frame time. */
pfSync();
```

```
    /* Compute new view position. */
    t = pfGetTime();
    pfSetVec3(view.hpr, 0.0f, -10.0f, 0.0f);
    pfSetVec3(view.xyz, 0.0f,
        -2.0f * bsphere->radius,
         0.5f * bsphere->radius);
    pfChanView(chan, view.xyz, view.hpr);

    /* Initiate cull/draw for this frame. */
    pfFrame();
    }

    /* Terminate parallel processes and exit. */
    pfExit();

    return 0;
}


/*
 *    OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
static void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

2. \PTG\examples\OpenGVS\smoke\smoke.c

```
/*
 * smoke.c:   simple Performer program to demonstrate using
 *            pfutil smoke
 *
 * $Revision: 1.6 $ $Date: 1994/03/16 09:35:33 $
 *
 */


#include "../../../Current/include/pfToGVS.h"

void OpenPipeline (pfPipe *p);
void DrawChannel(pfChannel *chan, void *data);

 void DrawChannel (pfChannel *channel, void *data)
{

}

/*
 *    Usage() -- print usage advice and exit. This
 *      procedure is executed in the application process.
 */
static void
Usage (void)
{
    fprintf(stderr, "Usage: smoke file.ext ...\n");
    exit(1);
}


int
pfMain (int argc, char *argv[])
{

    pfScene     *scene;
    pfPipe      *p;

    pfNode    *root;
    pfEarthSky    *esky;
    pfVec3    vec;
    pfuSmoke    *smoke;
    pfuSmoke    *fire;


    if (argc < 2)
    Usage();
```

```
/* Initialize Performer */
pfInit();

/* Use default multiprocessing mode based on number of
 * processors.
 */
pfMultiprocess(PFMP_DEFAULT);
bsphere = (pfSphere*) pfMalloc (sizeof(pfSphere),
   pfGetSharedArena());

/* Append to PFPATH additional standard directories where
 * geometry and textures exist
 */
pfFilePath("$GV_ROOT/gvm");

pfuInitSmokes();

/* Configure multiprocessing mode and start parallel
 * processes.
 */
pfConfig();

/* Read a single file, of any known type. */
if ((root = LoadFile(argv[1], NULL)) == NULL)
{
pfExit();
exit(-1);
}

/* Attach loaded file to a pfScene. */
scene = pfNewScene();
pfAddChild(scene, root);

/* determine extent of scene's geometry */
pfGetNodeBSphere (scene, bsphere);

/* Create a pfLightSource and attach it to scene. */
pfAddChild(scene, pfNewLSource());

/* Configure and open GL window */
p = pfGetPipe(0);
pfInitPipe(p, OpenPipeline);

/* Create and configure a pfChannel. */
chan = pfNewChan(p);
pfChanDrawFunc(chan, DrawChannel);
pfChanScene(chan, scene);
pfChanNearFar(chan, 1.0f, 10.0f * bsphere->radius);
pfChanFOV(chan, 45.0f, 0.0f);

esky = pfNewESky();
```

```
    pfESkyMode(esky, PFES_BUFFER_CLEAR, PFES_SKY_GRND);
    pfESkyAttr(esky, PFES_GRND_HT, -1.0f * bsphere->radius);
    pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
    pfESkyColor(esky, PFES_GRND_FAR, 0.3f, 0.1f, 0.0f, 1.0f);
    pfChanESky(chan, esky);

    pfSetVec3 (vec, 0.0f, 0.0f, 0.0f);

    fire = pfuNewSmoke();
    pfuSmokeType (fire, PFUSMOKE_FIRE);
    pfuSmokeOrigin (fire, vec, 0.25f * bsphere->radius);
    pfuSmokeMode (fire, PFUSMOKE_START);

    smoke = pfuNewSmoke();
    pfuSmokeType (smoke, PFUSMOKE_SMOKE);
    pfuSmokeOrigin (smoke, vec, 0.25f * bsphere->radius);
    pfuSmokeVelocity (smoke, 1.0f, 0.25f * bsphere->radius);
    pfuSmokeDensity (smoke, 1.f, 2.42f, 1.83f);
    pfuSmokeMode (smoke, PFUSMOKE_START);

    return 0;
}


/*
 *    OpenPipeline() -- create a GL window: set up the
 *      window system, IRIS GL, and IRIS Performer. This
 *      procedure is executed in the draw process (when
 *      there is a separate draw process).
 */
void
OpenPipeline (pfPipe *p)
{
    /* Open graphics window. */
    foreground();
    prefposition(100, 500, 100, 500);
    winopen("IRIS Performer");

    /* Configure window with reasonable defaults. */
    pfInitGfx(p);

    /* Create and apply a default material for those models
     * without one.
     */
    pfApplyMtl(pfNewMtl(pfGetSharedArena()));

    /* Create a default lighting model. */
    pfApplyLModel(pfNewLModel(pfGetSharedArena()));
}
```

a. *\PTG\examples\OpenGVS\smoke\ptgproj.c*

```
/*****************************************************************
*
*  ptgproj.c --
*
*       NPS Performer to OpenGVS Project Simulation Loop File
*
*****************************************************************/

#include "../../../Current/include/pfToGVS.h

 /*
 * GV_user_proc is automatically called once by GVS
 * every frame during system run-time. The Performer simulation loop
 * must be copied to this function, as is, after globals are ID'd
 * with a prefix. See PTG Users Manual.
 * f - 5feb96
 */
int GV_user_proc( void )
{


/*
** Put simulation loop in here.
**
**
*/
    pfCoord    view;

    /* Go to sleep until next frame time. */
    pfSync();

    /* Compute new view position. */
    t = pfGetTime();
    pfSetVec3(view.hpr, 0.0f, -10.0f, 0.0f);
    pfSetVec3(view.xyz, 0.0f,
        -2.0f * bsphere->radius,
         0.5f * bsphere->radius);
    pfChanView(chan, view.xyz, view.hpr);

    /* Initiate cull/draw for this frame. */
    pfFrame();


    /* Terminate parallel processes and exit. */

/*******************   End Simulation Loop    ************/
    return G_SUCCESS;

}
```

b. \PTG\examples\OpenGVS\smoke\ptgproj.h

```
/* ptgproj.h - > globals for the user_proc loop go here */
#ifndef __PTGPROJ_H
#define __PTGPROJ_H


__Globalutils__  float        t;
__Globalutils__  pfSphere     *bsphere;
__Globalutils__  pfChannel    *chan;

#endif
```

APPENDIX D
PROTOTYPE DISPLAY MANAGER FOR OPEN NPSNET APPLICATION ARCHITECTURE

This appendix contains a partial listing of a Proposed Display Manager for Open NPSNET source code that is compatible on both SGI and Window NT workstations. The prototype main is Virtual Reality Network (VRNET). A brief description of each of the header and source files are listed below.

1. **vrnet.cpp** -- This is the main program for a new, platform-independent version of NPSNET - follow-on work to the thesis project. (implemented here as vrnet for a PC)

2. **displMan.h** -- Display Manager header for the VRNET project.

3. **displMan.cpp** -- Display Manager class for the VRNET project.

4. **display.cpp** -- Display class for the VRNET project. Displays handle OpenGVS graphics pipeline stuff and scene graphs.

5. **events.cpp** -- User input functions for theVRNET project. Keyboard/mouse inputs change texture/object settings, and camera viewpoint to demo world.

A. SOURCE CODE LISTING

1. vrnet.c

```
/*****************************************************************
*
*  vrnet.c --  F.Free/J.Borrego
*
*  This is the main program for a new, platform-independent
*  version of NPSNET - follow-on work to the thesis project.
*  (implemented here as vrnet for a PC)
*
*  Start the separate managers, and loop until done.
*
*  25 July 1996
*
*****************************************************************/

#include <stdlib.h>

#include "displMan.h"

* Anticipate also needing other managers:
#include "entityMan.h"
#include "networkMan.h"
#include "inputMan.h"
*/

/* main - Platform independent main for new NPSNET
 *    architecture prototype. Main's only job is to
 *    start all the managers & loop for events.
 *    As of July, only the WinNT/OpenGVS display
 *    manager is implemented.
 * f - 29 july 96
 */

int main( int argc, char *argv[] )
{
    int done = 0;

    /* these are the managers for the new architecture */
    DisplayManagerClass* displayManager;
    /*
    InputManagerClass*   inputManager;
    NetworkManagerClass* networkManager;
    EntityManagerClass*  entityManager;
    */

    displayManager = new DisplayManagerClass( argc, argv );
```

```
    /* Test all managers for "done-ness" */
    while( !done )
    {
       displayManager->updateDisplay();

       done = displayManager->getStatus();  // Is window OK?
    }

    delete displayManager;

    return (EXIT_SUCCESS);
}
```

2. displMan.h

```
/*****************************************************************
*
*  displMan.h --  F.Free/J.Borrego
*
*       Display Manager header for the vrnet project,
*       July 96
*
*****************************************************************/
#ifndef __Globalutils__
#define __Globalutils__ extern
#endif

#ifndef __DISPLMAN_H
#define __DISPLMAN_H

#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <time.h>
#include <math.h>

#include <g_stdlib.h>
#include <gv_sys.h>
#include <gv_user.h>
#include <gv_chn.h>
#include <g_consts.h>
#include <gv.h>
#include <g_timer.h>
#include <gv_txr.h>
#include <gifts/tools/sts/stsgfx.h>
#include <g_math.h>
#include <g_gen.h>
#include <g_sys.h>

#include "tmse.h"


#define MAX_NUMBER_OF_ENTITIES 250



int GV_demo_sys( int argc, char ** argv );

enum ModelTypeEnum { not_assigned, terrain, hind, ah64,
    f16, m1, v22, vjeep, t62, hughes_500, blimp };

struct EntityStruct {
    int           entityID;
    ModelTypeEnum modelType;
```

```
    GV_Obi          entity;
    G_Vector3       velocity;
    G_Position      position;
    G_Rotation      rotation;
};


class DisplayClass {

   public:
      DisplayClass();                 // constructor
      ~DisplayClass(){}               // destructor
      void update();                  // update model info
      void process();                 // process model info
      int getNumOfDefEntities()
         { return numOfDefEntities; }

      // These are for the un-OO architecture prototype demo
      EntityStruct entityBuffer;
      void addEntity();
      void deleteEntity();



   private:
      GV_Fbf            fbf;          /* A GVS frame buffer    */
      GV_Camera         camera;       /* A GVS camera          */
      GV_Channel        channel;      /* A GVS channel         */
      GV_Scene          scene;        /* A GVS scene           */
      GV_Light          sun;          /* A GVS light           */
      GV_Obd  terrainDef, hindDef,    /* Some geom def names   */
         ah64Def, f16Def, m1Def,
         v22Def, vjeepDef, t62Def,
       hughes_500Def, blimpDef;

     EntityStruct entityList[MAX_NUMBER_OF_ENTITIES];
     int numOfDefEntities;
};


class DisplayManagerClass {

    public:
      DisplayManagerClass(int argc, char *argv[]);  // constructor
      ~DisplayManagerClass();                       // destructor
      void updateDisplay();                  // update model info
     int getStatus();                       // has display been killed?
      DisplayClass * display;

    private:
```

```
};

/* In other than this prototype demo, display should not be
 * global. The DM alone talks to it's display.
 * That'll work when the EM, NM, etc are written, and correct
 * message passing is implemented.
 */
__Globalutils__ DisplayClass * display;

#endif
```

3. displMan.cpp

```
/*************************************************************************
*
*  displMan.cpp --  F.Free/J.Borrego
*
*       Display Mnager class for the vrnet project,
*       July 96
*
*************************************************************************/
#define __Globalutils__

#include "displMan.h"


/* GV_user_init is automatically called once during system
 * initialization
 */

int GV_user_init( void )
{
    display = new DisplayClass();

    return G_SUCCESS;
}


/* Simple GVS run-time stuff for rotate project;
 * GV_user_proc is automatically called once by GVS as the default
 * callback every frame during system run-time.
 */
int GV_user_proc( void )
{

    display->update();  // entity/scene stuff
    display->process();  // display (camera view point) stuff

    return G_SUCCESS;
}

int GV_demo_sys( int argc, char ** argv )
{

    G_sys_set_units( 1.0, G_SYS_UNITS_METERS ) ;

    /* User routine to parse the Unix command line */
    GV_user_parse_cmd( argc, argv );

    /* Establish project specific callbacks */
    GV_sys_set_callback_init( GV_user_init );
```

```
    GV_sys_set_callback_proc( GV_user_proc );
    GV_sys_set_callback_shutdown( GV_user_shutdown );

    /* Initialize OpenGVS */
    GV_sys_init();

    GV_sys_set_mode( GV_SYS_MODE_RUNTIME );

    return (G_SUCCESS);
}


/* DisplayManagerClass constructor -
 *    Since this class is for a WinNT/OpenGVS specific
 *    display manager for an otherwise platform independent
 *    NPSNET program, the constructor does all the graphics
 *    resource allocation and windowing stuff.
 *    f - july96
 */
DisplayManagerClass::DisplayManagerClass( int argc, char* argv[] )
{
   if ( G_FAILURE == GV_demo_sys( argc, argv ) )
      exit(1) ;
   return;
}

DisplayManagerClass::~DisplayManagerClass()
{
   delete display;
   GV_sys_shutdown();
}

void DisplayManagerClass::updateDisplay()
{
    GV_sys_proc();
}

int DisplayManagerClass::getStatus()
{
   GV_Sys_mode system_mode;

   const int done = 1;

   GV_sys_inq_mode( &system_mode) ;
   return (system_mode == GV_SYS_MODE_RUNTIME) ? !done : done;
}
```

4. display.cpp

```
/*****************************************************************
*
*  display.cpp --  F.Free/J.Borrego
*
*       Display class for the vrnet project,
*       Displays handle OpenGVS graphics pipeline stuff
*       and scene graphs.
*       28 July 96
*
*****************************************************************/
#include "displMan.h"

/* DisplayClass constructor -
 *    Since this class is for a WinNT/OpenGVS specific
 *    display for an otherwise platform independent
 *    NPSNET program, the constructor does all the graphics
 *    resource allocation and windowing stuff.
 *    f - july96
 */
DisplayClass::DisplayClass()
{
    // Create graphics resources
    GV_fbf_create( &fbf );
    GV_chn_create( &channel );
    GV_cam_create( &camera );
    GV_scn_create( &scene );
    GV_lsr_create( &sun );

   // Configure channel (window)
   char initName[] = "VRNET - Prototype NPSNET Architecture";
   GV_Viewport normalized_viewport = {0.0, 1.0, 0.0, 1.0};
   GV_Rgba erase_color = {0.0, 1.0, 1.0, 1.0};
   float yon = 10000.0;
   float hither = 1.0;
   GV_chn_set_name( channel, initName );
   GV_chn_set_erase_color( channel, &erase_color );
   GV_chn_set_viewport( channel, &normalized_viewport );
   GV_chn_set_clip_near( channel, hither );
   GV_chn_set_clip_far( channel, yon );

   // Place camera's initial position/rotation  (ChanView)
   G_Position flycampos = { 0.0,  50.0, 50.0 };
   G_Rotation flycamrot = { -15.0 * G_DEG_TO_RAD, 0.0, 0.0 };
   GV_cam_set_position( camera, 0, &flycampos );
   GV_cam_set_rotation( camera, 0, &flycamrot );

  // Nice-to-have frame rate stats
  GV_Obi stats;
    stats = stsgfx_instance();
```

```
    GV_scn_add_object( scene, stats );
    GV_obi_set_name( stats, "STATS" );

    // Connect graphics resources
    GV_scn_add_light( scene, sun );
    GV_chn_set_camera( channel, camera );
    GV_chn_set_scene( channel, scene );
    GV_fbf_add_channel( fbf, channel );

    // Initialize entity list
    for(int ix = 0; ix < MAX_NUMBER_OF_ENTITIES; ix++)
    {
       entityList[ix].entityID = -1;
       entityList[ix].modelType = not_assigned;
       entityList[ix].velocity.x = entityList[ix].velocity.y =
           entityList[ix].velocity.z = 0;
    }

    // No entities created yet
    numOfDefEntities = 0;

   // Import geometry that _may_ be used
   GV_cmd_service("import file=$GV_ROOT/gvm/amusement.gvm
    name=terrainFile");          //terrain.flt

   GV_obd_inq_by_name("terrainFile", &terrainDef);
   GV_cmd_service("import file=$GV_ROOT/gvm/hind.gvm name=hindFile");
   GV_obd_inq_by_name("hindFile", &hindDef);
   GV_cmd_service("import file=$GV_ROOT/gvm/ah64.gvm name=ah64File");
   GV_obd_inq_by_name("ah64File", &ah64Def);
   GV_cmd_service("import file=$GV_ROOT/gvm/f16.gvm name=f16File");
   GV_obd_inq_by_name("f16File", &f16Def);
   GV_cmd_service("import file=$GV_ROOT/gvm/m1.gvm name=m1File");
   GV_obd_inq_by_name("m1File", &m1Def);
   GV_cmd_service("import file=$GV_ROOT/gvm/v22.gvm name=v22File");
   GV_obd_inq_by_name("v22File", &v22Def);
   GV_cmd_service("import file=$GV_ROOT/gvm/vjeep.gvm
      name=vjeepFile");
   GV_obd_inq_by_name("vjeepFile", &vjeepDef);
   GV_cmd_service("import file=$GV_ROOT/gvm/t62.gvm name=t62File");
   GV_obd_inq_by_name("t62File", &t62Def);
   GV_cmd_service("import file=$GV_ROOT/gvm/hughes_500.gvm
      name=hughes_500File");
   GV_obd_inq_by_name("hughes_500File", &hughes_500Def);

   // During devel, start with no textures - toggle w/ t or T
   GV_sys_set_attr_state( GV_SYS_STATE_TEXTURE, G_OFF);

}
```

```
void DisplayClass::update()              // devel stuff to look at
{
    // These inputs will come from the EM to DM later
        if(entityBuffer.entityID == 1)
              addEntity();
        if(entityBuffer.entityID == -1)
              deleteEntity();

    // Reset buffer
        entityBuffer.entityID = 0;

}
void DisplayClass::addEntity()
{
    numOfDefEntities++;
    entityList[numOfDefEntities].entityID = numOfDefEntities;
    entityList[numOfDefEntities].modelType = entityBuffer.modelType;

    // Big if (or case) to choose entity from list of possible models.
    // don't delete this one
    if (entityList[numOfDefEntities].modelType == terrain)
       GV_obi_instance(terrainDef,
          &entityList[numOfDefEntities].entity);
    else if (entityList[numOfDefEntities].modelType == hind)
      GV_obi_instance( hindDef, &entityList[numOfDefEntities].entity
);

    entityList[numOfDefEntities].velocity = entityBuffer.velocity;
    entityList[numOfDefEntities].rotation = entityBuffer.rotation;
    entityList[numOfDefEntities].position = entityBuffer.position;

    GV_scn_add_object( scene, entityList[numOfDefEntities].entity );

    cout << "a add" << numOfDefEntities << endl;
}

void DisplayClass::deleteEntity()
{
    if( numOfDefEntities )
    {
       GV_scn_remove_object( scene,
          entityList[numOfDefEntities].entity );
       GV_obi_free( entityList[numOfDefEntities--].entity );
    }
    cout << "a delete" << numOfDefEntities << endl;
}


void DisplayClass::process()
{
 int IDNumber;
```

```
// Update postion of moving objects
for ( int iz = 0; iz < numOfDefEntities; iz++ )
{
    IDNumber = iz + 1;
    if( (entityList[IDNumber].velocity.x > .1) ||
        (entityList[IDNumber].velocity.y > .1) ||
        (entityList[IDNumber].velocity.z > .1) )
   {  // No, this is not right, but it demos where the entity update
      // is displayed.
      entityList[IDNumber].position.x +=
         entityList[IDNumber].velocity.x;
      entityList[IDNumber].position.y +=
         entityList[IDNumber].velocity.y;
      entityList[IDNumber].position.x +=
         entityList[IDNumber].velocity.z;
      GV_obi_set_position( entityList[IDNumber].entity,
          &entityList[IDNumber].position );
    }
}

// Allow user to fly through the scene
{
     G_Vector2 cursor;
     GV_Channel chn_focus;
     float x_in, z_in;
     GV_chn_inq_input_focus( &chn_focus);
     if (chn_focus = channel)
     {
        static int base = 0;
        G_Position position;
        G_Rotation angles;

        GV_chn_inq_cursor_position( channel, &cursor );
        GV_cam_inq_position( camera, base, &position );
        GV_cam_inq_rotation( camera, base, &angles );
        tmse_move( &cursor, ((GV_Geometry)entityList[0].entity),
              &position, &angles );
         x_in = position.x;
         z_in = position.z;
        GV_cam_set_position_x( camera, base, x_in );
        GV_cam_set_position_z( camera, base, z_in );
        GV_cam_set_rotation( camera, base, &angles );
     }
  }
}
```

5. events.cpp

```
/*****************************************************************
*
*  events.cpp --  F.Free/J.Borrego
*
*       User input functions for the VRNET project.
*
*  Keyboard/mouse inputs change texture/object settings, and camera
*  viewpoint to demo world.
*
*****************************************************************/
#include "displMan.h"
#include "events.h"
#include <gifts/tools/sts/stsgfx.h>
#include <g_sys.h>
#include <gv_sys.h>
#include <gifts/tools/sts/stspvt.h>

/*static void queue_devices(void);*/

#define basecam 0



/*****************************************************************
*
*  event_add_entity & event_delete_entity
*     --  These functions _simulate_ the
*     event callbacks that the EntityManager would make to the
*     DisplayManager in the Open NPSNET Architecture. The completion
*     of that, and the other managers are follow-on work to the paper
*     by R.E. Barker, and the thesis by J. Borrego & F. Free. The
*     device callbacks should be replaced by EM callbacks possibly to
*     shared memory.
*
*****************************************************************/
void event_add_entity( int kcode )
{
    const G_Vector3 startVel = { 1.0, 1.0, -1.0 };
    G_Vector3 terrainVel = { 0.0, 0.0, 0.0 };
    G_Position position = {0.0,0.0,0.0};
    G_Rotation rotation ={0.0,0.0,0.0};
    static int first = 1;

    // First, add terrain - later add hinds.

    display->entityBuffer.entityID = 1;
    display->entityBuffer.modelType =
        ( display->entityBuffer.entityID == first ) ? terrain : hind;
```

```
    display->entityBuffer.velocity =
        ( display->entityBuffer.entityID == first ) ? terrainVel :
          startVel;
    display->entityBuffer.position = position;
    display->entityBuffer.rotation = rotation;

    first = 0;
}

void event_delete_entity( int kcode )
{
   // delete last entity
    display->entityBuffer.entityID = -1;
}


/*********************************************************************
*
*  event_translate_camera --  fly the camera along the axes x/y/z
*
*********************************************************************/

void event_translate_camera( int key )
{
   static float dx = 4.0;
   static float dy = 4.0;
   static float dz = 4.0;

   GV_Channel channel;
   (void)key; /* Not used: Avoid compiler warning */

   /* Get the channel with input focus, else the first one defined by
    * user
    */
   GV_chn_inq_input_focus( &channel );
   if( !channel )
      GV_chn_inq_first( &channel );

   if( channel )
   {
   /* Get the current camera for this channel */
      GV_Camera camera;
      GV_chn_inq_camera( channel, &camera );
      if ( camera )
      {
         G_Position position ;

         GV_cam_inq_position( camera, basecam, &position );

         switch ( key )
         {
```

```
            case 'x':
               if (position.x >= -2000)
                  position.x -= dx;
               break;

            case 'y':
              if (position.y >= -1)
                  position.y -= dy;
              break;

            case 'z':
               if (position.z >= -2000)
                  position.z -= dz;
               break;

            case 'X':
               if (position.x <= 2000)
                  position.x += dx;
               break;

            case 'Y':
               if (position.y <= 1000)
                  position.y += dy;
               break;

            case 'Z':
               if (position.z <= 2000)
                  position.z += dz;
               break;

   }

      GV_cam_set_position( camera, basecam, &position );
      }
   }
}


/*****************************************************************
*
*  event_rotate_camera -- slewing up/down or left/right
*
*****************************************************************/

void event_rotate_camera( int key )
{
   static float drx = (G_PI/180.0) * (2.0);
   static float dry = (G_PI/180.0) * (2.0);
   static float drz = (G_PI/180.0) * (2.0);

   GV_Channel channel;
```

```
    (void)key; /* Not used: Avoid compiler warning */

    /* Get the channel with input focus, else the first one defined by
     * user
     */

    GV_chn_inq_input_focus( &channel );
    if( !channel )
       GV_chn_inq_first( &channel );

    if( channel )
    {
         /* Get the current camera for this channel */
       GV_Camera camera;
       GV_chn_inq_camera( channel, &camera );
       if ( camera )
       {
          G_Rotation rotation;
          GV_cam_inq_rotation( camera, basecam, &rotation );

        switch ( key )
        {
           case 'u': case 'U':
              rotation.x += drx;
              break;

           case 'l': case 'L':
              rotation.y += dry;
              break;

           case 'd': case 'D':
              rotation.x -= drx;
              break;

           case 'r': case 'R':
              rotation.y -= dry;
              break;

        }
        GV_cam_set_rotation( camera, basecam, &rotation );
       }
    }
}


void stsgfx_key_toggle_detail_mode( int key )
{
    (void) key ; /* Avoid compiler warning for unused variable */

    if (stsgendat.detail_mode == G_OFF)
       stsgendat.detail_mode = G_ON ;
```

```
    else
     stsgendat.detail_mode = G_OFF ;
}

void toggle_statistics( int kcode )
{
    GV_Obi s2;
    GV_Scene scene1;
    G_State state1;

    GV_scn_inq_by_name( "SCENE", &scene1 );
    GV_obi_inq_by_name( "STATS", &s2 );
    GV_scn_remove_object( scene1, s2 );
    GV_scn_add_object( scene1, s2 );
    printf("stats\n");
    state1 = stsgfx_inq_mode();
    stsgfx_set_mode( /*!state1*/ (state1 == G_ON) ? G_OFF : G_ON );
}


void event_toggle_textures( int kcode )
{
     G_State state;
     GV_sys_inq_attr_state( GV_SYS_STATE_TEXTURE, &state);
     state = state ? G_OFF : G_ON;
     GV_sys_set_attr_state( GV_SYS_STATE_TEXTURE, state);
}


/*****************************************************************
*
*  queue_devices --
*
* Queue devices... GVS supports old GL devices since OpenGL does not
*
*****************************************************************/
static void queue_devices(void)
{
    qdevice(LEFTMOUSE);
    qdevice(RIGHTMOUSE);

    G_key_set_callback( 't' , event_toggle_textures );
    G_key_set_callback( 'T' , event_toggle_textures );

    G_key_set_callback( 'x' , event_translate_camera );
    G_key_set_callback( 'X' , event_translate_camera );
    G_key_set_callback( 'y' , event_translate_camera );
    G_key_set_callback( 'Y' , event_translate_camera );
    G_key_set_callback( 'z' , event_translate_camera );
    G_key_set_callback( 'Z' , event_translate_camera );
```

```
    G_key_set_callback( 'u' , event_rotate_camera );
    G_key_set_callback( 'd' , event_rotate_camera );
    G_key_set_callback( 'l' , event_rotate_camera );
    G_key_set_callback( 'r' , event_rotate_camera );
    G_key_set_callback( 'U' , event_rotate_camera );
    G_key_set_callback( 'D' , event_rotate_camera );
    G_key_set_callback( 'L' , event_rotate_camera );
    G_key_set_callback( 'R' , event_rotate_camera );

    G_key_set_callback( 'I', toggle_statistics );
    G_key_set_callback( 'i', stsgfx_key_toggle_detail_mode );


    /* These are here only to test the vrnet program,
     *   a prototype for the Open NPSNET Architecture.
     *   In the final version, events to add & delete entities
     *     will come from the Entity Manager, but for now, since the
     *     design by the NPSNET RG is not complete, we use PC input
     *   devices to test the theory.
     *   For this demo of the theory of the new architecture,
     *   some not-very OO stuff is required. For example
     *   "display" should not be global: a "real" entity
     *    manager would talk directly to a display manager, not the
     *    display. But with this, you get the idea...
     */
    G_key_set_callback( 'a', event_add_entity );
    G_key_set_callback( 'A', event_add_entity );
    G_key_set_callback( 'b', event_delete_entity );
    G_key_set_callback( 'B', event_delete_entity );

}
/*********************************************************************
*
*  GV_user_events --
*
* Called during run-time whenever OpenGVS traps an event including
* X11, mouse, and keyboard events.
*
*********************************************************************/
int GV_user_events( int device, int event )
{
    int status = G_SUCCESS;
    static G_Boolean first_time = G_TRUE;
    GV_Channel channel;

    static float dry = (G_PI/180.0) * (2.0);
    GV_Camera camera;
    G_Rotation rotation;
    GV_chn_inq_input_focus( &channel );
    GV_chn_inq_camera( channel, &camera );
```

```
    GV_cam_inq_rotation( camera, basecam, &rotation );

    if( first_time )
    {
       queue_devices();
       first_time = G_FALSE;
    }

    /* Limit mouse button events to 1 per half-second */
    if( device == LEFTMOUSE || device == MIDDLEMOUSE
       || device == RIGHTMOUSE )
    {
       static float time_elapsed = 10.0;
       static float old_time = 0.0;
       double new_time;

       G_timer_inq_time( &new_time );
       time_elapsed += new_time - old_time;
       old_time = new_time;

       if( time_elapsed < 0.5 )
          return( G_SUCCESS );
       else
          time_elapsed = 0.0;
    }

    switch( device )
    {
/***************
* MOUSE EVENTS *
***************/
   case LEFTMOUSE:
   {
      rotation.y += dry;
      GV_cam_set_rotation( camera, basecam, &rotation );
      printf("lt mouse....\n");
      break;
   }

   case RIGHTMOUSE:
   {
      rotation.y -= dry;
      GV_cam_set_rotation( camera, basecam, &rotation );
      printf("right mouse....\n");
      break;
   }

   }

   return( status );
}
```

LIST OF REFERENCES

[BARK96] Barker, Randal, "Open NPSNET Architecture Overview," Unpublished, 1996.

[BRUT95] Brutzman, Donald, Macedonia, Michael, R., Zyda, Michael J., "Internetwork Infrastructure Requirements for Virtual Environments," Presented at the VRML Symposium, San Diego California, December 13-15 1995, Available at http://www.stl.nps.navy.mil/~brutzman/vrml/vrml-95.html, Internet.

[ELSA96] Elsa Incorporated, "ELSA GLoria Product Specification Guide," Homepage Available at http://www.elsa.com, Internet, 1996.

[GEMI96] Gemini Technology Corporation, Homepage, Available at http://www.gemtech.com, Internet, 1996.

[GEMP96] Gemini Technology Corporation, "OpenGVS Programming Guide," Version 4.0.1, Document Number GVH-PG-4.0.1, February 1996.

[HART94] Hartman, Jed, and Creek, Patricia, "Iris Performer™ Programming Guide," 1994.

[IDEA96] IDEAS International Pty. Limited Homepage, Available at http://www.ideasinternational.com/, Internet.

[INTE96] Intel Homepage, Available http://www.intel.com/procs/perf/icomp/icomp_paper/, Internet.

[KANE96] Kane, Jim and McDonough, John, "Lab Report: 18: Graphics Cards Quick on the Draw," Byte, February 1996, pp.142-151.

[NPSN96] NPSNET Research Group Home page, Available at: http://www.npsnet.cs.navy.mil/npsnet/, Internet.

[PERF94] Fischler, Sharon, Helman, Jim, Jones, Michael, Rohlf, John, Schaffer, Allan, Tanner, Christopher, "Iris Performer™ Reference Pages," 1994.

[SCHE95] Schechter, Joanne, "Accelerating 3D Graphics," *Computer Graphics World,* October 1995, pp. 35-42.

[USEL96] Uselton, Sam, Cox, Michael, Deering, Michael, Torborg, Jay, and Akeley, Kurt, "Graphic PCs will put Workstation Graphics in the Smithsonian," Presented at the ACM SIGGRAPHP Conference Panel, New Orleans, LA, August 3-10, 1996.

[WILL95] Willeby, Tandy, "3D Graphics Boards," *3D Design*, Miller Freeman, Volume 1, Number 1, 1995, pp. 66-71.

TRADEMARK INFORMATION

3Dlabs and GLINT are registered trademarks of 3Dlabs Inc. Ltd. GLINT 300 SX is a trademark of 3Dlabs, Inc.

AG300 and Accel Graphics are trademarks of Accel Graphics, Inc.

ELSA and GLoria are trademarks of ELSA Inc., and ELSA GmbH, Aachen (Germany).

OpenGVS is a trademark of Gemini Technology Corporation.

Intel, Pentium, and iCOMP are registered trademarks of Intel. Intel486, Intel386, and Intel486DX2 66 trademarks of Intel Corporation.

Microsoft, Windows 95, MS-DOS, and Visual C++ are registered trademarks of Microsoft Corporation. Windows NT is a trademark of Microsoft Corporation.

R4400 is a registered trademark of MIPS Technologies, Inc.

3Demon and Omnicomp are trademarks of Omnicomp Graphics Corporation.

S3 is a registered trademarks of S3, Inc. Vision968 is a trademarks of S3, Inc.

Silicon Graphics, IRIS, and OpenGL are registered trademarks of Silicon Graphics Inc. IRIS Performer, Performer, IRIS GL, GL, IRIX, Onyx, RealityEngine2 and Indigo2 IMPACT are trademarks of Silicon Graphics, Inc.

STB is a registered trademarks of STB Systems, Inc.

Java is a registered trademark of Sun Microsystems, Inc. Sun is a trademark of Sun Microsystems, Inc.

Intergraph is a registered trademark of Intergraph Corporation.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C46
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

6. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Dr. David R. Pratt, Assistant Professor 2
   Computer Science Department Code CS/Pr
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. John S. Falby, Lecturer 2
   Computer Science Department Code CS/FJ
   Naval Postgraduate School
   Monterey, CA 93943-5000

9. Major Frank Free, USMC 2
   370 Bergin Drive, Apt. E
   Monterey, CA 93940

10. LT Jaime Borrego, USN 2
    100 Brownell Circle
    Monterey, CA 93940